

29th Annual European Symposium on Algorithms

ESA 2021, September 6–8, 2021, Lisbon, Portugal
(Virtual Conference)

Edited by

Petra Mutzel

Rasmus Pagh

Grzegorz Herman



Editors

Petra Mutzel 

University of Bonn, Germany
petra.mutzel@cs.uni-bonn.de

Rasmus Pagh 

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

Grzegorz Herman 

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

ACM Classification 2012

Computing methodologies → Concurrent computing methodologies; Computing methodologies → Feature selection; Computing methodologies → Search methodologies; Mathematics of computing → Algebraic topology; Mathematics of computing → Approximation algorithms; Mathematics of computing → Discrete mathematics; Mathematics of computing → Distribution functions; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Graph enumeration; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Matroids and greedoids; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Permutations and combinations; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Random graphs; Mathematics of computing → Topology; Mathematics of computing → Trees; Networks → Network algorithms; Security and privacy → Pseudonymity, anonymity and untraceability; Theory of computation → Algorithm design techniques; Theory of computation → Algorithmic game theory; Theory of computation → Algorithmic mechanism design; Theory of computation → Approximation algorithms analysis; Theory of computation → Backtracking; Theory of computation → Cell probe models and lower bounds; Theory of computation → Computational complexity and cryptography; Theory of computation → Computational geometry; 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 → Dynamic programming; Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis; Theory of computation → Machine learning theory; Theory of computation → Models of computation; Theory of computation → Network flows; Theory of computation → Network optimization; Theory of computation → Online algorithms; Theory of computation → Packing and covering problems; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Pseudorandomness and derandomization; Theory of computation → Quantum complexity theory; Theory of computation → Quantum computation theory; Theory of computation → Random network models; Theory of computation → Rounding techniques; Theory of computation → Scheduling algorithms; Theory of computation → Semidefinite programming; Theory of computation → Shared memory algorithms; Theory of computation → Shortest paths; Theory of computation → Sparsification and spanners

ISBN 978-3-95977-204-4

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/978-3-95977-204-4>.

Publication date

September, 2021

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 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/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.2021.0

ISBN 978-3-95977-204-4

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*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Petra Mutzel, Rasmus Pagh, and Grzegorz Herman</i>	0:xi
Program Committees	
.....	0:xiii
List of External Reviewers	
.....	0:xv

Invited Talks

Network Planning and Routing Problems over Time: Models, Complexity and Algorithms	
<i>Lukas Glomb, Benno Hoch, Frauke Liers, and Florian Rösel</i>	1:1–1:3
A User Friendly Power Tool for Deriving Online Learning Algorithms	
<i>Aaron Roth</i>	2:1–2:1

Regular Papers

Bi-Objective Search with Bi-Directional A*	
<i>Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby</i>	3:1–3:15
A Unified Approach for All Pairs Approximate Shortest Paths in Weighted Undirected Graphs	
<i>Maor Akav and Liam Roditty</i>	4:1–4:18
The Voronoi Diagram of Rotating Rays With applications to Floodlight Illumination	
<i>Carlos Alegría, Ioannis Mantas, Evanthia Papadopoulou, Marko Savić, Hendrik Schrezenmaier, Carlos Seara, and Martin Suderland</i>	5:1–5:16
Parallel Computation of Combinatorial Symmetries	
<i>Markus Anders and Pascal Schweitzer</i>	6:1–6:18
Graph Connectivity and Single Element Recovery via Linear and OR Queries	
<i>Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna</i>	7:1–7:19
Fully Dynamic Set Cover via Hypergraph Maximal Matching: An Optimal Approximation Through a Local Approach	
<i>Sepehr Assadi and Shay Solomon</i>	8:1–8:18
The Randomized Competitive Ratio of Weighted k -Server Is at Least Exponential	
<i>Nikhil Ayyadevara and Ashish Chiplunkar</i>	9:1–9:11
Orienting (Hyper)graphs Under Explorable Stochastic Uncertainty	
<i>Euripidis Bampis, Christoph Dürr, Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlotter</i>	10:1–10:18
k -Distinct Branchings Admits a Polynomial Kernel	
<i>Jørgen Bang-Jensen, Kristine Vitting Klinkby, and Saket Saurabh</i>	11:1–11:15

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman



Leibniz International Proceedings in Informatics

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

Incremental Edge Orientation in Forests <i>Michael A. Bender, Tsvi Kopelowitz, William Kuszmaul, Ely Porat, and Clifford Stein</i>	12:1–12:18
k -Center Clustering with Outliers in the Sliding-Window Model <i>Mark de Berg, Morteza Monemizadeh, and Yu Zhong</i>	13:1–13:13
Incremental SCC Maintenance in Sparse Graphs <i>Aaron Bernstein, Aditi Dudeja, and Seth Pettie</i>	14:1–14:16
Lyndon Words Accelerate Suffix Sorting <i>Nico Bertram, Jonas Ellert, and Johannes Fischer</i>	15:1–15:13
Online Euclidean Spanners <i>Sujoy Bhore and Csaba D. Tóth</i>	16:1–16:19
Distant Representatives for Rectangles in the Plane <i>Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud</i>	17:1–17:18
Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles <i>Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck</i>	18:1–18:17
Synchronized Planarity with Applications to Constrained Planarity Problems <i>Thomas Bläsius, Simon D. Fink, and Ignaz Rutter</i>	19:1–19:14
Efficiently Approximating Vertex Cover on Scale-Free Networks with Underlying Hyperbolic Geometry <i>Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann</i>	20:1–20:15
Efficiently Computing Maximum Flows in Scale-Free Networks <i>Thomas Bläsius, Tobias Friedrich, and Christopher Weyand</i>	21:1–21:14
Asymptotically Optimal Welfare of Posted Pricing for Multiple Items with MHR Distributions <i>Alexander Braun, Matthias Buttkus, and Thomas Kesselheim</i>	22:1–22:16
Covert Computation in Staged Self-Assembly: Verification Is PSPACE-Complete <i>David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie</i>	23:1–23:18
An Instance-Optimal Algorithm for Bichromatic Rectangular Visibility <i>Jean Cardinal, Justin Dallant, and John Iacono</i>	24:1–24:14
Worst-Case Efficient Dynamic Geometric Independent Set <i>Jean Cardinal, John Iacono, and Grigorios Koumoutsos</i>	25:1–25:15
Balanced Crown Decomposition for Connectivity Constraints <i>Katrin Casel, Tobias Friedrich, Davis Issac, Aikaterini Niklanovits, and Ziena Zeif</i>	26:1–26:15
All-Pairs Shortest Paths for Real-Weighted Undirected Graphs with Small Additive Error <i>Timothy M. Chan</i>	27:1–27:9
Dynamic Colored Orthogonal Range Searching <i>Timothy M. Chan and Zhengcheng Huang</i>	28:1–28:13

ℓ_p -Norm Multiway Cut <i>Karthekeyan Chandrasekaran and Weihang Wang</i>	29:1–29:15
Faster Algorithms for Longest Common Substring <i>Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski</i>	30:1–30:17
Feature Cross Search via Submodular Optimization <i>Lin Chen, Hossein Esfandiari, Gang Fu, Vahab S. Mirrokni, and Qian Yu</i>	31:1–31:16
An FPT Algorithm for the Embeddability of Graphs into Two-Dimensional Simplicial Complexes <i>Éric Colin de Verdière and Thomas Magnard</i>	32:1–32:17
Efficient Sequential and Parallel Algorithms for Multistage Stochastic Integer Programming Using Proximity <i>Jana Cslovjcek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel</i>	33:1–33:14
Modular Counting of Subgraphs: Matchings, Matching-Splittable Graphs, and Paths <i>Radu Curticapean, Holger Dell, and Thore Husfeldt</i>	34:1–34:17
Minimum Common String Partition: Exact Algorithms <i>Marek Cygan, Alexander S. Kulikov, Ivan Mihajlin, Maksim Nikolaev, and Grigory Reznikov</i>	35:1–35:16
An Accelerated Newton–Dinkelbach Method and Its Application to Two Variables per Inequality Systems <i>Daniel Dadush, Zhuan Khye Koh, Bento Natura, and László A. Végh</i>	36:1–36:15
Faster 3-Coloring of Small-Diameter Graphs <i>Michał Dębski, Marta Piecyk, and Paweł Rzążewski</i>	37:1–37:15
Stability Yields Sublinear Time Algorithms for Geometric Optimization in Machine Learning <i>Hu Ding</i>	38:1–38:19
Data Structures Lower Bounds and Popular Conjectures <i>Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová</i>	39:1–39:15
Approximation Schemes for Bounded Distance Problems on Fractionally Treewidth-Fragile Graphs <i>Zdeněk Dvořák and Abhiruk Lahiri</i>	40:1–40:10
Modular and Submodular Optimization with Multiple Knapsack Constraints via Fractional Grouping <i>Yaron Fairstein, Ariel Kulik, and Hadas Shachnai</i>	41:1–41:16
Differentially Private Algorithms for Graphs Under Continual Observation <i>Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost</i>	42:1–42:16
Experimental Comparison of PC-Trees and PQ-Trees <i>Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter</i>	43:1–43:13

Boundary-Sensitive Approach for Approximate Nearest-Neighbor Classification <i>Alejandro Flores-Velazco and David M. Mount</i>	44:1–44:15
Compression by Contracting Straight-Line Programs <i>Moses Ganardi</i>	45:1–45:16
Space Efficient Two-Dimensional Orthogonal Colored Range Counting <i>Younan Gao and Meng He</i>	46:1–46:17
Computing the 4-Edge-Connected Components of a Graph in Linear Time <i>Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas</i>	47:1–47:17
Deep Multilevel Graph Partitioning <i>Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier</i>	48:1–48:17
Faster $(1 + \epsilon)$ -Approximation for Unsplittable Flow on a Path via Resource Augmentation and Back <i>Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese</i>	49:1–49:15
Quantum Sub-Gaussian Mean Estimator <i>Yassine Hamoudi</i>	50:1–50:17
Improved Approximation Algorithms for Tverberg Partitions <i>Sariel Har-Peled and Timothy Zhou</i>	51:1–51:15
Near-Linear-Time, Optimal Vertex Cut Sparsifiers in Directed Acyclic Graphs <i>Zhiyang He, Jason Li, and Magnus Wahlström</i>	52:1–52:14
Closing the Gap for Single Resource Constraint Scheduling <i>Klaus Jansen and Malin Rau</i>	53:1–53:15
Certified Approximation Algorithms for the Fermat Point and n -Ellipses <i>Kolja Junginger, Ioannis Mantas, Evanthia Papadopoulou, Martin Suderland, and Chee Yap</i>	54:1–54:19
Parameterized Algorithms for Diverse Multistage Problems <i>Leon Kellerhals, Malte Renken, and Philipp Zschoche</i>	55:1–55:17
Fast and Space-Efficient Construction of AVL Grammars from the LZ77 Parsing <i>Domènec Kempa and Ben Langmead</i>	56:1–56:14
Convex Drawings of Hierarchical Graphs in Linear Time, with Applications to Planar Graph Morphing <i>Boris Klemz</i>	57:1–57:15
QCSP on Reflexive Tournaments <i>Benoît Larose, Petar Marković, Barnaby Martin, Daniël Paulusma, Siani Smith, and Stanislav Žitný</i>	58:1–58:15
Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing <i>Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu</i>	59:1–59:17
Telescoping Filter: A Practical Adaptive Filter <i>David J. Lee, Samuel McCauley, Shikha Singh, and Max Stein</i>	60:1–60:18

Finding an Approximate Mode of a Kernel Density Estimate <i>Jasper C.H. Lee, Jerry Li, Christopher Musco, Jeff M. Phillips, and Wai Ming Tai</i>	61:1–61:19
An Efficient Reduction of a Gammoid to a Partition Matroid <i>Marilena Leichter, Benjamin Moseley, and Kirk Pruhs</i>	62:1–62:13
Efficient Algorithms for Least Square Piecewise Polynomial Regression <i>Daniel Lokshтанov, Subhash Suri, and Jie Xue</i>	63:1–63:15
Bidirectional String Anchors: A New String Sampling Mechanism <i>Grigorios Loukides and Solon P. Pissis</i>	64:1–64:21
The Visibility Center of a Simple Polygon <i>Anna Lubiw and Anurag Murty Naredla</i>	65:1–65:14
Extension of Additive Valuations to General Valuations on the Existence of EFX <i>Ryoga Mahara</i>	66:1–66:15
FPT and FPT-Approximation Algorithms for Unsplittable Flow on Trees <i>Tomás Martínez-Muñoz and Andreas Wiese</i>	67:1–67:15
A Simple Algorithm for Graph Reconstruction <i>Claire Mathieu and Hang Zhou</i>	68:1–68:18
Generalized Max-Flows and Min-Cuts in Simplicial Complexes <i>William Maxwell and Amir Nayyeri</i>	69:1–69:16
Hypersuccinct Trees – New Universal Tree Source Codes for Optimal Compressed Tree Data Structures and Range Minima <i>J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild</i>	70:1–70:18
Determining 4-Edge-Connected Components in Linear Time <i>Wojciech Nadara, Mateusz Radecki, Marcin Smulewicz, and Marek Sokółowski</i>	71:1–71:15
Isomorphism Testing Parameterized by Genus and Beyond <i>Daniel Neuen</i>	72:1–72:18
Restricted t -Matchings via Half-Edges <i>Katarzyna Paluch and Mateusz Wasylkiewicz</i>	73:1–73:17
Beating Random Assignment for Approximating Quantum 2-Local Hamiltonian Problems <i>Ojas Parekh and Kevin Thompson</i>	74:1–74:18
Additive Sparsification of CSPs <i>Eden Pelleg and Stanislav Žitný</i>	75:1–75:15
Faster Deterministic Modular Subset Sum <i>Krzysztof Potępa</i>	76:1–76:16
Hardness of Detecting Abelian and Additive Square Factors in Strings <i>Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba</i>	77:1–77:19

On Approximate Compressions for Connected Minor-Hitting Sets	
<i>M. S. Ramanujan</i>	78:1–78:16
Restricted Adaptivity in Stochastic Scheduling	
<i>Guillaume Sagnol and Daniel Schmidt genannt Waldschmidt</i>	79:1–79:14
A General Framework for Enumerating Equivalence Classes of Solutions	
<i>Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinimeri</i>	80:1–80:14
Engineering MultiQueues: Fast Relaxed Concurrent Priority Queues	
<i>Marvin Williams, Peter Sanders, and Roman Dementiev</i>	81:1–81:17
Evidence for Long-Tails in SLS Algorithms	
<i>Florian Wörz and Jan-Hendrik Lorenz</i>	82:1–82:16

Preface

This volume contains the extended abstracts selected for presentation at ESA 2021, the 29th European Symposium on Algorithms. The event was planned by an organizing committee from University of Lisbon, Portugal as part of ALGO 2021. Due to the COVID-19 pandemic, ALGO was organized as an on-line event, rather than as a physical or hybrid event as originally planned, September 6–8, 2021.

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 that also address 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 2021, 320 papers were submitted, 274 for Track A and 46 for Track B (these are the counts after track changes and withdrawals). Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews. The program committees selected 80 papers for inclusion in the program, 68 from track A and 12 from track B, yielding an acceptance rate of about 1/4 for both tracks.

The presentations of the accepted papers together with two invited talks resulted in a very exciting program. Frauke Liers presented recent work on the complexity of algorithms for network optimization tasks with a timing component, and Aaron Roth about a user-friendly framework for deriving online learning algorithms.

The European Association for Theoretical Computer Science (EATCS) sponsored best paper and best student paper awards. The best paper award for track A was given to Zhiyang He, Jason Li and Magnus Wahlström for the paper *Near-linear-time, Optimal Vertex Cut Sparsifiers in Directed Acyclic Graphs*. For track B, the best paper award was given to Simon D. Fink, Matthias Pfretzschner and Ignaz Rutter for the paper *Experimental Comparison of PC-Trees and PQ-Trees*. 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 Wojciech Nadara, Mateusz Radecki, Marcin Smulewicz and Marek Sokołowski for the paper *Determining 4-edge-connected Components in Linear Time*. For track B, the best student paper award was given to Florian Wörz and Jan-Hendrik Lorenz for the paper *Evidence for Long-Tails in SLS Algorithms*.

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 the nearly 500 external reviewers who assisted the program committees in the evaluation process. Special thanks go to the organizing committee, who helped us with the organization of the conference.

■ Program Committees

Track A (Design and Analysis) Program Committee

- Akanksha Agrawal, Indian Institute of Technology Madras
- Andrea Lincoln, University of California, Berkeley
- Andrew McGregor, University of Massachusetts, Amherst
- Asaf Levin, Technion - Israel Institute of Technology
- Benjamin Doerr, École Polytechnique
- Chaitanya Swamy, University of Waterloo
- Clément Canonne, University of Sydney
- Debarati Das, University of Copenhagen
- Édouard Bonnet, École normale supérieure de Lyon
- Elias Koutsoupas, University of Oxford
- Eric Price, University of Texas at Austin
- Frances Rosamond, University of Bergen
- Gautam Kamath, University of Waterloo
- Gonzalo Navarro, University of Chile
- Gramoz Goranci, University of Toronto
- Guy Kortsarz, Rutgers University
- Harald Räcke, Technical University Munich
- Huacheng Yu, Princeton University
- Ilan Reuven Cohen, Bar Ilan University
- Ilias Diakonikolas, University of Wisconsin, Madison
- Irina Kostitsyna, Eindhoven University of Technology
- Jakub Łącki, Google Research
- Jason Li, Carnegie Mellon University
- Jeff M. Phillips, University of Utah
- Johannes Fischer, Technical University of Dortmund
- Justin Thaler, Georgetown University
- Laxman Dhulipala, Massachusetts Institute of Technology
- Maarten Löffler, Utrecht University
- Marco Molinaro, Pontifical Catholic University of Rio de Janeiro
- Marvin Künnemann, Max Planck Institute for Informatics
- Melanie Schmidt, University of Cologne
- Michał Pilipczuk, University of Warsaw
- MohammadTaghi HajiAghayi, University of Maryland
- Naveen Garg, Indian Institute of Technology Delhi
- Omri Weinstein, Columbia University and Hebrew University
- Paloma T. Lima, University of Bergen
- Pierre Fraigniaud, Université de Paris and CNRS
- Ran Duan, Tsinghua University
- Rasmus Kyng, ETH Zurich
- Rasmus Pagh (Chair), University of Copenhagen
- Richard Peng, Georgia Institute of Technology and University of Waterloo
- Riko Jacob, IT University of Copenhagen
- Rossano Venturini, University of Pisa

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman



Leibniz International Proceedings in Informatics

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

- Simon Apers, Université libre de Bruxelles and CWI Amsterdam
- Talya Eden, Massachusetts Institute of Technology
- Tobias Christiani, Norwegian University of Science and Technology
- Tobias Friedrich, Hasso Plattner Institute and University of Potsdam
- Yusuke Kobayashi, Kyoto University
- Zachary Friggstad, University of Alberta

Track B (Engineering and Applications) Program Committee

- Petra Berenbrink, University of Hamburg
- Therese Biedl, University of Waterloo
- Gerth Stølting Brodal, Aarhus University
- Sanjeeb Dash, IBM T. J. Watson Research Center
- Kathrin Hanauer, University of Vienna
- Susana Ladra, University of A Coruña
- Leo Liberti, CNRS & Institut Polytechnique de Paris
- Ivana Ljubic, ESSEC Business School of Paris
- Andrea Marino, University of Florence
- Henning Meyerhenke, Humboldt-Universität zu Berlin
- Petra Mutzel (Chair), University of Bonn
- Manuel Penschuck, Goethe University Frankfurt
- Cynthia Phillips, Sandia National Laboratories
- Liam Roditty, Bar-Ilan University
- Daniel Schmidt, University of Bonn
- Nodari Sitchinava, University of Hawaii at Manoa
- Blair D. Sullivan, University of Utah
- Yihan Sun, University of California, Riverside
- Christos Zaroliagis, University of Patras

■ List of External Reviewers

Amir Abboud	Olivier Bodini
Andreas Abels	Greg Bodwin
Faisal Abu-Khzam	Fritz Bökler
Peyman Afshani	Vincenzo Bonnici
Deniz Ağaoğlu	Jan van den Brand
Sara Ahmadian	Franz Brandenburg
Hee-Kap Ahn	Milutin Brankovic
Susanne Albers	Boris Brimkov
Josh Alman	Karl Bringmann
Aris Anagnostopoulos	Frederik Brünig
Daniel Anderson	Kevin Buchin
Eugenio Angriman	Andrei Bulatov
Claudia Archetti	Laurent Bulteau
Diego Arroyuelo	Jonathan Bunton
Sepehr Assadi	Silvia Butti
Martin Aumüller	Jarosław Byrka
Pranjal Awasthi	Sergio Cabello
Kyriakos Axiotis	Yang Cai
Arturs Backurs	Parinya Chalermsook
Ainesh Bakshi	Yi-Jun Chang
Hideo Bannai	Vaggos Chatziafratis
Nikhil Bansal	Chandra Chekuri
Jérémy Barbay	Sitan Chen
Siddharth Barman	Andrew Childs
Frederique Bassino	Rajesh Chitnis
Jatin Batra	Man-Kwun Chiu
Pierre-Olivier Baudry	Rezaul Chowdhury
Ruben Becker	Andrea Clementi
Soheil Behnezhad	Christian Coester
Djamal Belazzougui	Vincent Cohen-Addad
Aleksandrs Belovs	Alessio Conte
Ioana Bercea	Alexander Conway
Kristóf Bérczi	Arjan Cornelissen
Lorenzo Beretta	Mattia D'Emidio
Jeremias Berg	Konrad K. Dabrowski
Sebastian Berndt	Yuval Dagan
Nico Bertram	Syamantak Das
Aditya Bhaskara	Sami Davies
Sayan Bhattacharya	Gianluca De Marco
Marcin Bieńkowski	Arnaud De Mesmay
Philip Bille	Ronald de Wolf
Davide Bilò	Oscar Defrain
Ahmad Biniaz	Argyrios Deligkas
Eric Blais	Olivier Devillers

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman



Leibniz International Proceedings in Informatics

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

Tamal Dey	Yu Gao
Diego Díaz-Domínguez	Shivam Garg
Patrick Dinklage	Paweł Gawrychowski
Joao F. Doriguello	Evangelia Gergatsouli
Michal Dory	Nima Ghanbari
David Doty	Prantar Ghosh
Jan Dreier	Archontia Giannopoulou
Lukas Drexler	Jacob Gilbert
Anne Driemel	Yuval Gitlitz
Aditi Dudeja	Petr Golovach
Bartłomiej Dudek	Alexander Golovnev
Zdeněk Dvořák	Florian Göttl
Matthijs Ebbens	Themistoklis Gouleakis
Yuval Efron	Dishant Goyal
Eduard Eiben	Fabrizio Grandoni
Friedrich Eisenbrand	Kasper Green Larsen
Marek Elias	Christoph Grunau
Robert Elsässer	Yan Gu
Yuval Emek	Joachim Gudmundsson
Jessica Enright	André L. P. Guedes
Alessandro Epasto	Anupam Gupta
David Eppstein	Sushmita Gupta
Leah Epstein	Akshay Gupta
Thomas Erlebach	Arash Haddadan
Guy Even	Koki Hamada
Tomer Ezra	Thekla Hamm
Rolf Fagerberg	Sariel Har-Peled
Matthew Fahrbach	Christopher Harshaw
Chenglin Fan	David Hartvigsen
Reza Fathi	Haleh Havvaei
Moran Feldman	Meng He
Amos Fiat	Qizheng He
Hendrik Fichtenberger	Monika Henzinger
Gabriele Fici	Danny Hermelin
Yuval Filmus	Illya Hicks
Kyle Fox	Dorit Hochbaum
Fabrizio Fra ti	Ivor van der Hoog
Borja Freire	Svein Høgemo
Vincent Froese	Christopher Hojny
José Fuentes-Sepúlveda	Alexandros Hollender
Takuro Fukunaga	Chien-Chung Huang
Radoslav Fulek	Minyi Huang
Eric Fusy	Shang-En Huang
Elisabeth Gaar	Sophie Huiberts
Kasimir Gabert	Christoph Hunkenschröder
Kshitij Gajjar	John Iacono
Arun Ganesh	Sharat Ibrahimpur
Robert Ganian	Ilias Iliadis

Tanmay Inamdar
 Shunsuke Inenaga
 Kazuo Iwama
 Chuzo Iwamoto
 Satoru Iwata
 Lars Jaffke
 Bart M. P. Jansen
 Artur Jež
 Łukasz Jež
 Shaofeng H.-C. Jiang
 Ce Jin
 Tomasz Jurdziński
 Dominik Kaaser
 Naonori Kakimura
 Sagar Kale
 Pritish Kamath
 Naoyuki Kamiyama
 Panagiotis Kanellopoulos
 Lawqueen Kanesh
 Iyad Kanj
 Sampath Kannan
 Ahmet Kara
 Adam Karczmarz
 Juha Kärkkäinen
 Sushrut Karmalkar
 Karthik C. S.
 Telikepalli Kavitha
 Vahideh Keikha
 Phillip Keldenich
 Hans Kellerer
 Dominik Kempa
 Thomas Kesselheim
 Sammy Khalife
 Shahbaz Khan
 Eun Jung Kim
 Evangelos Kipouridis
 Sándor Kisfaludi-Bak
 Ralf Klasing
 Jonathan Klawitter
 Linda Kleist
 Peter Kling
 Fabian Klute
 Christine Klymko
 Marina Knittel
 Dušan Knop
 Tomohiro Koana
 Tomasz Kociumaka
 Burak Kocuk

Christian Komusiewicz
 Joshua Marc Könen
 Spyros Kontogiannis
 Tsvi Kopelowitz
 Dominik Köppl
 Martin Koutecký
 Elias Koutsoupias
 László Kozma
 Artur Kraska
 Ravishankar Krishnaswamy
 Danny Krizanc
 Ariel Kulik
 Janardhan Kulkarni
 Akash Kumar
 Amit Kumar
 Ravi Kumar
 Ron Kupfer
 Florian Kurpicz
 Tsz Chiu Kwok
 O-Joung Kwon
 Bundit Laekhanukit
 Manuel Lafond
 Alexandra Lassota
 Silvio Lattanzi
 Brian Lavallee
 Philip Lazos
 Hung Le
 Erik Jan van Leeuwen
 Roie Levin
 Fei Li
 Jerry Li
 Ray Li
 Tongyang Li
 Yi Li
 Wei-Kai Lin
 Hsiang-Hsuan Liu
 Mingmou Liu
 Quanquan Liu
 Yang Liu
 William Lochet
 Henri Lotze
 Anna Lubiw
 Troels Bjerre Lund
 Kelin Luo
 Jayakrishnan Madathil
 Florent Madelaine
 Anil Maheshwari
 Diptapriyo Majumdar

Veli Mäkinen
David Manlove
Pasin Manurangsi
Giovanni Manzini
Tomáš Masařík
Luke Mathieson
Jannik Matuschke
Simon Mauras
Charles McGuffey
Kitty Meeks
Nicole Megow
Arne Meier
Alexsander Melo
Arturo Merino
George Mertzios
Neeldhara Misra
Pranabendu Misra
Joseph Mitchell
Yosuke Mizutani
Tobias Mömke
Fabrizio Montecchiani
Stefano Moretti
Jamie Morgenstern
Benjamin Moseley
Amer Mouawad
David Mount
Ramin Mousavi
Marcin Mucha
Moritz Mühlenthaler
Anirbit Mukherjee
Anish Mukherjee
Sagnik Mukhopadhyay
Alexander Munteanu
Kishen N Gowda
Wojciech Nadara
Viswanath Nagarajan
Seffi Naor
Meghana Nasre
Jesper Nederlof
Ofer Neiman
Yakov Nekrich
Stefan Neumann
Calvin Newport
André Nichterlein
Benjamin Niedermann
Alexander Noe
Krzysztof Nowicki
André Nusser

Zeev Nutov
Johannes Obenaus
Carlos Ochoa
Shunhao Oh
Yoshio Okamoto
Karolina Okrasa
Dániel Oláh
Jan Olkowski
Federico Olmedo
Tim Oosterwijk
Sebastian Ordyniak
Wolfgang Ost
Sang-il Oum
Kenta Ozeki
Sourabh Palande
Katarzyna Paluch
Irene Parada
Nikos Parotsidis
Konstantinos Parsopoulos
Merav Parter
Ori Parzanchevski
Kanstantsin Pashkovich
Matthew Patitz
John Peebles
Mercedes Pelegrin
Seth Pettie
Marc Pfetsch
Jeff Phillips
Solon Pissis
Adam Polak
Nicola Prezza
Maximilian Probst
Simon Puglisi
Kent Quanrud
Yihui Quek
Tomasz Radzik
Chrysanthi Raftopoulou
Saladi Rahul
Benjamin Raichel
Cyrus Rashtchian
Malin Rau
Dror Rawitz
Bhaskar Ray Chaudhury
Daniel Reichman
Knut Reinert
Mohsen Rezapour
Javiel Rojas-Ledesma
Jérémie Roland

Christian Rosenke	Andy Sun
Marc Roth	Xiaorui Sun
Aviad Rubinstein	Andrew Sutton
Alexander Russell	Svend Christian Svendsen
Kunihiko Sadakane	Zoya Svitkina
Kostis Sagonas	Alexander Svozil
Hamed Saleh	Kenjiro Takazawa
Samitha Samaranayake	Zihan Tan
Alberto Santini	Jing Tang
Thatchaphol Saranurak	Erin Taylor
Srinivasa Rao Satti	David Tench
Thomas Sauerwald	Sharma V. Thankachan
David Saulpic	Takeshi Tokuyama
Saket Saurabh	Tigran Tonoyan
Jonathan Scarlett	Noam Touitou
Till Schäfer	Hung Tran
Markus Schepers	Meng-Tsung Tsai
Kevin Schewior	Konstantinos Tsakalidis
Arne Schmidt	Kostas Tsichlas
Jon Schneider	Charalampos Tsourakakis
Patrick Schnider	Christos Tzamos
Chris Schwiegelshohn	Seeun William Umboh
Saeed Seddighin	Jorge Urrutia
Xiangkun Shen	Erlend Raa Vågset
Thomas Shermer	Ali Vakilian
Jessica Shi	Rob van Stee
Francesco Silvestri	Daniel Vaz
Kirill Simonov	Luca Versari
Vikrant Singhal	José Verschae
Friedrich Slivovsky	Alexandre Vigny
Jack Snoeyink	Hoa Vu
Christian Sohler	Tal Wagner
Marek Sokolowski	Magnus Wahlström
Mehdi Soleimanifar	Erik Waingarten
Kiril Solovey	Bartosz Walczak
Frank Sommer	Romain Wallon
José A. Soto	Yipu Wang
Bruce Spang	Yiqiu Wang
Renan Spencer Trindade	Justin Ward
Frits Spieksma	Julian Wargalla
Andreas Spillner	Alexander Wein
Joachim Spoerhase	Nicole Wein
Jonathan Spreer	Philip Wellnitz
Max Springer	Andreas Wiese
Frank Staals	Sebastian Wild
Sabine Storandt	Lisa Wilhelmi
Torstein Strømme	David Williamson
Warut Suksompong	Alexander Wires

Michał Włodarczyk
Marcin Wrochna
Xiaowei Wu
Mingyu Xiao
Liding Xu
Yinzhan Xu
Yutaro Yamaguchi
Hadi Yami
Yongjie Yang
Penghui Yao

Yitong Yin
Tom van der Zanden
Iyar Zaks
Peter Zeman
Fred Zhang
Jialin Zhang
Weijie Zheng
Peilin Zhong
Samson Zhou
Shuchen Zhu

Network Planning and Routing Problems over Time: Models, Complexity and Algorithms

Lukas Glomb ✉ 🏠

Department of Data Science, Department Mathematik,
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Benno Hoch ✉ 🏠

Department of Data Science, Department Mathematik,
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Frauke Liers ✉ 🏠

Department of Data Science, Department Mathematik,
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Florian Rösel ✉ 🏠

Department of Data Science, Department Mathematik,
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Abstract

In this invited contribution for ESA 2021, we will study the complexity of and algorithms for network optimization tasks with a timing component. They occur, for example, in planning or routing problems that need to be solved repeatedly over time. Typically, already simplified versions of such problems are NP-hard. In addition, the instances typically are too large to be solved straight-forwardly on a time-expanded graph. After an introduction into the area, we state the problem of determining best possible non-stop trajectories in a network that are not allowed to cross at any point in time. For simplified settings, polynomial-time solution approaches are presented whereas already for restricted settings the problems are shown to be NP-hard. When moving to more complex and more realistic settings as they occur, for example, in determining non-stop disjoint trajectories for a set of aircraft, we present heuristic algorithms that adaptively refine coarse disjoint trajectories in the timing dimension. In order to be able to solve the non-stop disjoint trajectories problem over time, the method is integrated in a rolling-horizon algorithm. We present computational results for realistic settings. Motivated by the fact that rolling-horizon approaches are often applied in practice without knowledge on the quality of the obtained solutions, we study this problem from an abstract point of view. In fact, we more abstractly analyze the solution quality of general rolling-horizon algorithms for optimization tasks that show a timing component. We apply it to different planning problems. We end by pointing out some challenges and possibilities for future research.

2012 ACM Subject Classification Theory of computation; Mathematics of computing → Discrete mathematics

Keywords and phrases network problems over time, rolling-horizon, complexity, approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.1

Category Invited Talk

Related Version *Full Version*: http://www.optimization-online.org/DB_HTML/2020/09/8015.html

Full Version: http://www.optimization-online.org/DB_HTML/2020/05/7809.html

Funding The author acknowledges financial support by BMWi through the projects HOTRUN, FKZ 20E1720B, and OPs-TIMAL, FKZ 20X1711L.



© Lukas Glomb, Benno Hoch, Frauke Liers, and Florian Rösel;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 1; pp. 1:1–1:3

Leibniz International Proceedings in Informatics



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

1 Summary

In this contribution, we will introduce several network optimization tasks that contain a timing component. Such tasks often occur in planning or routing problems that need to be solved repeatedly over time. Already simplified such network optimization tasks are NP-hard. In addition, realistic instances are typically too large to be solved in an integrated fashion. On the other hand, straight-forward decomposition approaches typically lead to bad solution quality.

We first introduce the class of optimization problems that are considered here. We will then more concretely show complexity results for several settings together with algorithms and computational evaluations. We start by introducing the task of determining best possible non-stop trajectories in a network that are not allowed to cross at any point in time. For simplified settings, polynomial-time solution approaches are presented. Those settings include instances with unit traversal times on the line or grids and meshes, where all commodities have the same destination. For several only slightly more complicated settings the problems are shown to be NP-hard. For example, if time intervals to start the trajectories are given, the problem on the line is hard. On the mesh with arbitrary arc costs the problem is not fixed-parameter tractable in the number of commodities, even if traversal times are unit. More details can be found in [5].

We then move on to more complex and more realistic settings as they occur, for example, in determining non-stop disjoint trajectories for a set of aircraft [4, 7, 8]. We focus on the surroundings of airports where both the planning of conflict-free trajectories as well as the determination of runway schedules are crucial and challenging tasks. In current practice, the resulting continuous and discrete optimization problems are often solved sequentially. In this work, we develop an integrated optimization model for conflict-free multi-aircraft trajectory planning and runway scheduling. We use a space-time discretization and model conflict-free trajectories by an integer linear program that is designed to provide optimal, piecewise-linear reference trajectories and a runway scheduling for multiple aircraft. Even for moderately sized instances, a sufficiently detailed representation of 3D-airspace and time leads to huge models, which cannot be treated by current hard- and software. To overcome this issue, we develop an iterative adaptive-refinement algorithm. Starting from an optimal solution in a coarse discretization, the algorithm re-optimizes trajectories in a neighborhood of the current solution with a higher resolution. The method is integrated into a rolling-horizon approach. The latter repeatedly restricts trajectory determinations within a (sliding) time window. Computational results on realistic instances illustrate the computational efficiency of our approach.

The relevance of the rolling-horizon approach has been demonstrated by applying it to solve a large variety of practical optimization problems, e.g. as in [6] or [1]. Hence it is very interesting to know theoretical properties of the rolling-horizon approach, which have been investigated, e.g. in [2] or [9]. We complement these works in [3]. Hence, we analyze the solution quality of general rolling-horizon algorithms that are applied to multi-period optimization problems with a timing component.

We demonstrate that the solution quality of the standard rolling-horizon procedure can be arbitrarily low considering our general problem setting. We thus adapt the general rolling-horizon procedure such that statements on the quality of solutions obtained from adapted rolling-horizon algorithms can be made.

On the practical side, we present computational results on lot-sizing problems in production planning as well as on tail-assignment problems in aircraft management. The latter assigns available aircraft to specific flights in a best possible way. It can be shown that huge

instances can be solved quickly with an almost negligible loss in solution quality by only a few percent. We end by pointing out some challenges and possibilities for future research.

References

- 1 Bernardetta Addis, Giuliana Carello, Andrea Grosso, and Elena Tànfani. Operating room scheduling and rescheduling: a rolling horizon approach. *Flexible Services and Manufacturing Journal*, 28(1-2):206–232, 2016.
- 2 James C Bean and Robert L Smith. Conditions for the existence of planning horizons. *Mathematics of Operations Research*, 9(3):391–401, 1984.
- 3 Lukas Glomb, Frauke Liers, and Florian Rösel. A rolling-horizon approach for multi-period optimization. *European Journal of Operational Research*, 2021. doi:10.1016/j.ejor.2021.07.043.
- 4 Benedikt Grüter, Matthias Bittner, Matthias Rieck, Johannes Diepolder, and Florian Holzapfel. Optimal sequencing in atm combining genetic algorithms and gradient based methods to a bilevel approach. In *ICAS 30th International Congress of the International Council of the Aeronautical Sciences*, 2016.
- 5 Benno Hoch, Frauke Liers, Sarah Neumann, and Francisco Javier Zaragoza Martinez. The non-stop disjoint trajectories problem, 2020.
- 6 Mohammad Mohammadi, SMT Fatemi Ghomi, Behrooz Karimi, and S Ali Torabi. Rolling-horizon and fix-and-relax heuristics for the multi-product multi-level capacitated lotsizing problem with sequence-dependent setups. *Journal of Intelligent Manufacturing*, 21(4):501–510, 2010.
- 7 Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *American Control Conference, 2002. Proceedings of the 2002*, volume 3, pages 1936–1941. IEEE, 2002.
- 8 Johannes Schmidt and Armin Fügenschuh. A two-time-level model for mission and flight planning of an inhomogeneous fleet of unmanned aerial vehicles. Technical report, BTU Cottbus, 2021.
- 9 Suresh Sethi and Gerhard Sorger. A theory of rolling horizon decision making. *Annals of Operations Research*, 29(1):387–415, 1991.

A User Friendly Power Tool for Deriving Online Learning Algorithms

Aaron Roth ✉

University of Pennsylvania, Philadelphia, PA, USA

Abstract

In this talk, we overview a simple and user friendly framework developed in [4] that can be used to derive online learning algorithms in a number of settings. In the core framework, at every round, an adaptive adversary introduces a new game, consisting of an action space for the learner, an action space for the adversary, and a vector valued objective function that is concave-convex in every coordinate. The learner and the adversary then play in this game. The learner’s goal is to play so as to minimize the maximum coordinate of the cumulative vector-valued loss. The resulting one-shot game is not concave-convex, and so the minimax theorem does not apply. Nevertheless we give a simple algorithm that can compete with the setting in which the adversary must announce their action first, with optimally diminishing regret.

We demonstrate the power of our simple framework by using it to derive optimal bounds and algorithms across a variety of domains. This includes no regret learning: we can recover optimal algorithms and bounds for minimizing external regret, internal regret, adaptive regret, multigroup regret, subsequence regret, and permutation regret in the sleeping experts setting. It also includes (multi)calibration [2] and related notions: we are able to recover recently derived algorithms and bounds for online adversarial multicalibration [1], mean conditioned moment multicalibration [3], and prediction interval multivalidity [1]. Finally we use it to derive a new variant of Blackwell’s Approachability Theorem, which we term “Fast Polytope Approachability”.

2012 ACM Subject Classification Theory of computation → Machine learning theory

Keywords and phrases Online Learning, Multicalibration, Multivalidity, Blackwell Approachability

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.2

Category Invited Talk

References

- 1 Varun Gupta, Christopher Jung, Georgy Noarov, Mallesh M Pai, and Aaron Roth. Online multivalid learning: Means, moments, and prediction intervals. *arXiv preprint arXiv:2101.01739*, 2021.
- 2 Ursula Hébert-Johnson, Michael Kim, Omer Reingold, and Guy Rothblum. Multicalibration: Calibration for the (computationally-identifiable) masses. In *International Conference on Machine Learning*, pages 1939–1948. PMLR, 2018.
- 3 Christopher Jung, Changhwa Lee, Mallesh M Pai, Aaron Roth, and Rakesh Vohra. Moment multicalibration for uncertainty estimation. In *Conference on Learning Theory*. PMLR, 2021.
- 4 Georgy Noarov, Mallesh Pai, and Aaron Roth. Online multiobjective minimax optimization and applications. *Manuscript*, 2021. [arXiv:2108.03837](#).



© Aaron Roth;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics




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

Bi-Objective Search with Bi-Directional A*

Saman Ahmadi¹  

Department of Data Science and AI, Monash University, Clayton, Australia
CSIRO Data61, Canberra, Australia

Guido Tack 

Department of Data Science and AI, Monash University, Clayton, Australia

Daniel Harabor 

Department of Data Science and AI, Monash University, Clayton, Australia

Philip Kilby 

CSIRO Data61, Canberra, Australia

Abstract

Bi-objective search is a well-known algorithmic problem, concerned with finding a set of optimal solutions in a two-dimensional domain. This problem has a wide variety of applications such as planning in transport systems or optimal control in energy systems. Recently, bi-objective A*-based search (BOA*) has shown state-of-the-art performance in large networks. This paper develops a bi-directional and parallel variant of BOA*, enriched with several speed-up heuristics. Our experimental results on 1,000 benchmark cases show that our bi-directional A* algorithm for bi-objective search (BOBA*) can optimally solve all of the benchmark cases within the time limit, outperforming the state of the art BOA*, bi-objective Dijkstra and bi-directional bi-objective Dijkstra by an average runtime improvement of a factor of five over all of the benchmark instances.

2012 ACM Subject Classification Computing methodologies → Search methodologies; Theory of computation → Shortest paths

Keywords and phrases Bi-objective search, heuristic search, shortest path problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.3

Related Version *Previous Version:* <https://arxiv.org/abs/2105.11888>

Funding Research at Monash University is supported by the Australian Research Council (ARC) under grant numbers DP190100013 and DP200100025 as well as a gift from Amazon.

1 Introduction

Bi-objective search aims at finding a set of non-dominated, Pareto-optimal solutions in a domain with two objectives [2]. It has a wide range of real-world applications, such as planning routes for maritime transportation based on both the fuel consumption and the total risk of the vehicle route [18], or energy efficient paths for electric vehicles with arrival time considerations [13]. When the underlying system is a network, the problem is finding a set of paths between two points that are not dominated by other solution paths.

A comparison of traditional approaches to the bi-objective one-to-all shortest path problem, such as the label correcting algorithm in [15], the label setting approach in [6], and the adaptation of a near shortest path procedure in [1], was presented in [11]. These label-based approaches have been extended in several recent papers. A generalisation of Dijkstra’s algorithm and its bi-directional counterpart (for the one-to-one variant) to the bi-objective problem was presented in [12] by utilising the pruning strategies of [5] to avoid

¹ Corresponding Author



expanding unpromising paths during the search. The results show that the state-of-the-art bi-objective Dijkstra algorithm can outperform the bounded label setting approach in [10] and the depth-first search-based *Pulse* algorithm in [5] on large-size instances.

Another recent work on point-to-point bi-objective search is the Bi-Objective A* search scheme (BOA*) in [17]. BOA* is a standard A* heuristic search that leverages the fast dominance checking procedure of [9] for multi-objective search. In contrast to eager dominance checking approaches, as in [12], BOA* lazily postpones dominance checking for newly generated nodes until their expansion. The experimental results in [17] on a set of large instances show that the efficient dominance checking helps BOA* to perform better than the bi-objective Dijkstra algorithm of [12] and other best-first search approaches such as the label-setting multi-objective search NAMOA* of [8] and its improved version with a dimensionality reduction technique called NAMOA*_{dr} [9].

In this paper, we present Bi-Objective Bi-directional A* (BOBA*), a bi-directional extension of the BOA* algorithm that is easy to parallelise, uses different objective orders and includes several new heuristics to speed up the search. Our experiments on a set of 1,000 large test cases from the literature show that BOBA* can solve all of the cases to optimality, outperforming the state-of-the-art algorithms in both runtime and memory requirement.

2 Background and Notation

For a directed bi-objective graph $G = (S, E)$ with a finite set of states S and a set of edges $E \subseteq S \times S$, the point-to-point bi-objective search problem is to find the set of Pareto-optimal solution paths from $start \in S$ to $goal \in S$ that are not dominated by any solution for both objectives. Every edge $e \in E$ has two non-negative attributes accessed via the cost function $\mathbf{cost} : E \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$. A path is a sequence of states $s_i \in S$ with $i \in \{1, \dots, n\}$. The cost of path $p = \{s_1, s_2, s_3, \dots, s_n\}$ is then defined as the sum of corresponding attributes on all the edges constituting the path as $\mathbf{cost}(p) = \sum_{i=1}^{n-1} \mathbf{cost}(s_i, s_{i+1})$. Following the standard notation in the heuristic search literature, we define our search objects to be *nodes*. A node x is a tuple that contains a state $s(x) \in S$; a value $\mathbf{g}(x)$ which measures the cost of a concrete path from the *start* state to state $s(x)$; a value $\mathbf{f}(x)$ which is an estimate of the cost of a complete path from *start* to *goal* via $s(x)$; and a reference $parent(x)$ which indicates the parent of node x . We perform a systematic search by *expanding* nodes in best-first order. Each expansion operation *generates* a set of successor nodes, each denoted $Succ(s(x))$, which are added into an *Open* list. The *Open* list sorts the nodes according to their \mathbf{f} -values in an ascending order, for the purpose of further expansion.

As with other A*-based algorithms, we compute \mathbf{f} -values using a consistent and admissible heuristic function $\mathbf{h} : S \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$ [7]. In other words, $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s)$ where $\mathbf{h}(s)$ is a lower bound on the cost of paths from state s to *goal*. Moreover, in bi-objective search, the cost function has two components which means that every (boldface) cost function is a tuple, eg. $\mathbf{f} = (f_1, f_2)$ or $\mathbf{h} = (h_1, h_2)$ and all operations are considered element-wise.

► **Definition 1.** A heuristic function \mathbf{h} is consistent if we have $\mathbf{h}(s) \leq \mathbf{cost}(s, t) + \mathbf{h}(t)$ for every edge $(s, t) \in E$. It is also admissible iff $\mathbf{h}(s) \leq \mathbf{cost}(p)$ for every $s \in S$ and the optimal path p from state s to the goal state.

► **Definition 2.** For every pair of nodes (x, y) associated with the same state $s(x) = s(y)$, node y is dominated by x if we have $g_1(x) < g_1(y)$ and $g_2(x) \leq g_2(y)$ or if we have $g_1(x) = g_1(y)$ and $g_2(x) < g_2(y)$. Node x weakly dominates y if $g_1(x) \leq g_1(y)$ and $g_2(x) \leq g_2(y)$.

Bi-objective A*. The Bi-Objective A* (BOA*) algorithm [17] first obtains its heuristic function h using two basic one-to-all searches on the reversed graph. BOA* can then establish lower bounds on the cost of complete paths or f -values using the admissible heuristic h . Although either of the two objectives can potentially play the key role in the bi-objective setting, standard BOA* usually chooses the first objective in the (f_1, f_2) order. The search then expands all the promising nodes based on their cost estimates so as to ensure the node with the lexicographically smallest f -value is explored first. The algorithm terminates when there is no node in *Open* while keeping all the non-dominated nodes associated with the *goal* state in the solution set *Sol*. The main steps of the standard BOA* algorithm can be found in Algorithm 2, scripted with normal line numbers (without asterisk *) in black.

► **Theorem 3.** *BOA* computes a set of cost-unique Pareto-optimal solution paths [17].*

BOA* utilises an efficient strategy to check nodes for their dominance, originally employed in [9] for multi-objective search. The idea is simple yet powerful. Let us assume A* explores the graph in the (f_1, f_2) order, that is, nodes are extracted based on their f_1 -value in order (with tie-breaking on f_2 -values). Meanwhile, x and y are two nodes associated with the same state or $s(x) = s(y)$ in the *Open* list where x is going to be expanded first, i.e., we have $f_1(x) \leq f_1(y)$. Since both nodes have used the same heuristic value as $h_1(s(x)) = h_1(s(y))$ to determine their cost estimate f_1 , we can conclude $g_1(x) \leq g_1(y)$. Therefore, the second node will be dominated by the first node if $g_2(x) \leq g_2(y)$ as shown in [9] in detail. BOA* takes advantage of this dimension reduction technique by systematically keeping track of the g_2 -value of the last non-dominated node using $g_2^{min}(s(x))$ via line 11 of Algorithm 2.

BOA* can also prune some of the dominated nodes during the expansion with a similar reasoning via line 28 of Algorithm 2. This is done by comparing the newly generated node of a state and the last expanded node of the state against their secondary costs g_2 . Furthermore, BOA* prunes unpromising nodes based on their cost estimate to the *goal* state, which is known as pruning by bound. Given $g_2^{min}(goal)$ as the upper bound of the secondary cost, partial paths will be pruned if the cost estimate of their complete paths to *goal* on g_2 is greater than that of the last solution already stored in $g_2^{min}(goal)$. Interested readers are referred to the standard BOA* algorithm in [17] for the detailed proof discussion.

Challenges. Lazy dominance checking in BOA* slows down the operations in the *Open* list and consumes more space. In contrast to the costly linear dominance checking approach where new nodes are checked against all of the previously generated nodes associated with a state before their insertion into the *Open* list, BOA* may add a node for which we have an unexpanded dominant node in *Open*. Thus, the search generates more nodes (using extra memory), and the *Open* list will inevitably be longer. Moreover, BOA* is only able to search the graph in one direction and with a specific objective ordering, whereas there can be cases with better performance on the reverse objective ordering as shown in [17]. Our preliminary experiments also reveal that searching backwards (from *goal* to *start*) may lead to significant improvements in the overall runtime. There are also some inefficiencies in BOA* which can be addressed with extra considerations. As an example, for the simple graph in Figure 2, BOA* needs to expand all intermediate states for each individual solution, despite the fact that some of them are not offering any alternative (non-dominated) path to *goal* (eg. s_2).

3 Bi-objective Bi-directional A* Search

Recent improvements in bi-directional heuristic search have introduced new techniques to reduce the number of necessary node expansions, such as Near-Optimal Bi-directional Search in [16] and Dynamic Vertex Cover Bi-directional Search in [14]. Given the single-objective

■ **Algorithm 1** Bi-Objective Bidirectional A* (BOBA*) High-level.

Input: A problem instance $(G, \mathbf{cost}, s_{start}, s_{goal})$
Output: A set of cost-unique Pareto-optimal solutions

```

1 do in parallel
2    $h'_1, ub'_2 \leftarrow \mathbf{cost}\text{-bounded A}^*$  from  $s_{start}$  to  $s_{goal}$  on  $G$  in  $(f_1, f_2)$  order
3    $h_2, ub_1 \leftarrow \mathbf{cost}\text{-bounded A}^*$  from  $s_{goal}$  to  $s_{start}$  on  $\text{Reversed}(G)$  in  $(f_2, f_1)$  order
4 do in parallel
5    $h'_2, ub'_1 \leftarrow \mathbf{cost}\text{-bounded A}^*$  from  $s_{start}$  to  $s_{goal}$  on  $G$  in  $(f_2, f_1)$  order
6    $h_1, ub_2 \leftarrow \mathbf{cost}\text{-bounded A}^*$  from  $s_{goal}$  to  $s_{start}$  on  $\text{Reversed}(G)$  in  $(f_1, f_2)$  order
7 do in parallel
8    $Sol \leftarrow \text{BOA}^*_{\text{enh}}$  for  $(G, \mathbf{cost}, s_{start}, s_{goal})$  with heuristics  $(\mathbf{h}, \mathbf{ub}, \mathbf{h}')$  in  $(f_1, f_2)$  order
9    $Sol' \leftarrow \text{BOA}^*_{\text{enh}}$  for  $(\text{Rev}(G), \mathbf{cost}, s_{goal}, s_{start})$  with heuristics  $(\mathbf{h}', \mathbf{ub}', \mathbf{h})$  in  $(f_2, f_1)$  order
10 return  $Sol + Sol'$ 

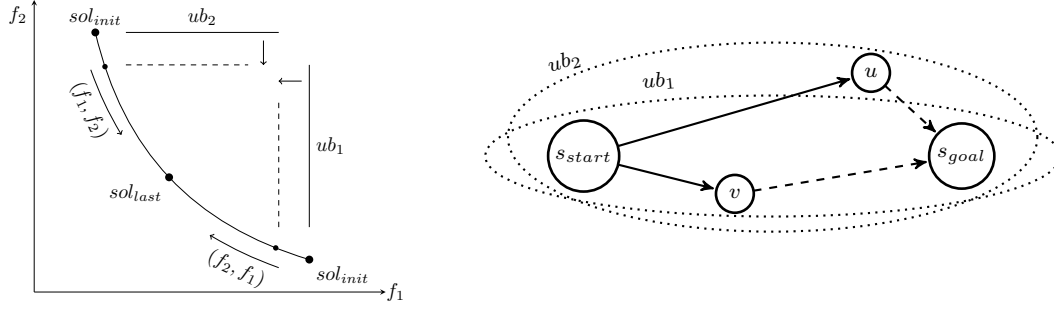
```

nature of the conventional shortest path problem, none of the existing front-to-end or front-to-front algorithms can practically tackle the bi-objective shortest path problem without incorporating necessary modifications. Moreover, those algorithms are not necessarily efficient for the bi-objective search as obtaining the solutions' cost would no longer be possible in $O(1)$ time. In the conventional bi-objective setting where both searches work on the same objective, every state offers a set of non-dominated nodes (partial paths) in each direction, and handling frontier collisions (obtaining all of the complete *start-goal* joined paths of the state) would be an exhaustive process which can outweigh the speed-up achieved by expanding fewer nodes. Our preliminary experiments also confirmed that the conventional front-to-end bi-directional search with an efficient partial paths coupling approach can potentially generate fewer nodes but shows poor performance compared to the unidirectional search scheme BOA*.

We now present our contributions to the problem by explaining our Bi-Objective Bi-directional A* search (BOBA*). BOBA* employs two complementary (enhanced) unidirectional BOA* to search the solution space in both (forward and backward) directions with different objective orders $((f_1, f_2)$ and $(f_2, f_1))$. Therefore, since the algorithm does not perform partial paths coupling, we do not need to handle frontier collisions. In other words, each uni-directional BOA* is allowed to explore the entire graph towards the opposite end for each individual solution. The high level structure of BOBA* is given in Algorithm 1. BOBA* first obtains the preliminary heuristics and then performs two individual searches that explore the graph in both directions concurrently. The output will then be the aggregation of solutions found in each search routine. To avoid searching for the same cost-optimum paths in both directions, BOBA* always chooses different orders for each direction. Figure 1(Left) depicts the way Pareto-optimal solutions are found based on two searches in the two orders. Initial solutions (sol_{init}) at both ends are typically the minimum cost paths already obtained via the heuristic searches for each objective. These cost-optimum paths can also initialise the global upper bounds (ub_1, ub_2) needed by the pruning by bound strategies in BOA*. The upper bounds are updated (always decreasing) during the search every time a valid solution is found, and sol_{last} is the last solution for which we have had $f_1 < ub_1$ and $f_2 < ub_2$.

► **Definition 4.** For every state $s \in S$, $\mathbf{ub}(s)$ is the upper bound on **cost** of complementary paths from state s to goal, eg., $ub_1(s)$ denotes the upper bound on $cost_1$.

► **Definition 5.** A path/node/state x is invalid if its estimated costs $\mathbf{f}(x)$ are not in the search global upper bounds (ub_1, ub_2) , i.e., x is invalid if $f_1(x) \geq ub_1$ or $f_2(x) \geq ub_2$.



■ **Figure 1** Left: Objective orders, bounds and Pareto-optimal solutions. Right: Schematic of states outside or inside of upper bounds. State u is out of bounds for f_1 and will be discarded in f_2 search.

3.1 Preliminary Heuristics

BOBA* requires both lower and upper bounds on the costs of complementary paths for each direction via four individual searches. In each search, we calculate a state's upper bound to be the cost of the optimum path using the non-primary objective. For example, the optimum path to state s for the first objective sets both $h_1(s)$ and $ub_2(s)$ (here $ub_2(s)$ is the cost of the path using the second objective). BOA* traditionally uses two runs of Dijkstra's algorithm to initialise lower bounds. For difficult cases, this initialisation time is usually outweighed by the main search time, but there can be simple cases where the total time of these heuristic searches dominates the main search time, especially in large instances. As a more efficient initialisation approach, we replace Dijkstra's algorithm with cost-bounded A* (or cost-bounded Dijkstra without heuristics), as formally stated in Lemma 6 and shown in lines 2-6 of Algorithm 1.

► **Lemma 6.** *The preliminary A* search on f_1 (or f_2) can terminate before expanding a state with $f_1 > ub_1$ (or $f_2 > ub_2$).*

Proof. Assume that a forward BOA* is intended and, therefore, the corresponding heuristics (via two backward searches) are required. If we start with two simple backward A* searches (one for each objective), each optimum *start-goal* path gives us two bounds as $(h_1(s_{start}), ub_2(s_{start}))$ and $(h_2(s_{start}), ub_1(s_{start}))$. Now, given $h_1(s_{start})$ and $h_2(s_{start})$ as the global lower bounds on f_1 and f_2 -values respectively, we will have $f_1 \geq h_1(s_{start})$ and $f_2 \geq h_2(s_{start})$ for every *start-goal* path. Therefore, any state with a cost estimate of $f_1 > ub_1(s_{start})$ in the A* search on the first objective, and similarly $f_2 > ub_2(s_{start})$ in the search on the second objective, will be dominated by one of the optimum solutions. On the other hand, since A* expands states in an increasing order of f -values, each heuristic search can terminate early with the first out-of-bound state, guaranteeing that all paths via unexplored states are already dominated. ◀

Algorithm 1 shows the parallel computation of all necessary heuristics in BOBA* in two phases. In the first phase (lines 2-3), we can execute our cost-bounded A* using any admissible heuristic for the primary objective (f_1 or f_2) and with tie-breaking on the secondary objective (f_2 or f_1). Note that the upper bounds are unknown prior to the searches in phase one, i.e., we initially have $ub_1 = ub_2 = \infty$, but we can update our global upper bounds as soon as we establish the optimal solution in each search. The initialisation step of BOBA* can be further improved for the heuristic searches in the opposite direction in phase two (lines 5-6). Once the necessary heuristics in one direction have been obtained, the heuristic search

in the opposite direction can use the lower bounds obtained from the first round as more informed heuristics. That is, the second phase of our cost-bounded A* searches are normally executed faster. Moreover, the opposite search in the second round can take advantage of the reduced search space resulting from the first round, delivering better quality heuristics without needing to expand already invalidated (out-of-bound) states. Lemma 8 states this technique more formally.

► **Example 7.** State v in Figure 1 (right) is within the upper bound of both objectives and will be expanded in the opposite direction. However, state u is observed out of bounds for the first objective (but within the bound of the second objective) and will then be discarded if it is going to be expanded in the second round of our heuristic searches. Note that violating at least one objective's upper bound is enough to mark nodes (or states) invalid.

► **Lemma 8.** *In the preliminary A* search on f_1 (or f_2), states with an estimated cost of $f_2 > ub_2$ (or $f_1 > ub_1$) are not part of any solution path.*

Proof. States with $f_2 > ub_2$ are dominated by the optimum path obtained for the first objective. This means that unexplored states with an estimated cost of $f_2 > ub_2$ are all invalid. Therefore, the following search on f_1 can ignore expanding such states knowing that no non-dominated solution can be found via invalid states. The same reasoning is valid for the reverse order. ◀

3.2 Bi-directional Search

BOBA* performs two enhanced BOA* concurrently, one from each direction. Algorithm 2 shows the details of our first enhanced BOA* algorithm used in BOBA* (forward search in the (f_1, f_2) order). Lines scripted in black are from the standard BOA* and the red lines with an asterisk (*) next to line numbers are our proposed enhancements. To be consistent with the BOA* notation, we obtain the latest global upper bounds from g_{min} values, i.e., we have $g_2^{min}(s_{goal}) = ub_2$ and $g_1^{min}(s_{start}) = ub_1$. This is because the forward search on (f_1, f_2) updates $g_2^{min}(s_{goal})$ for every solution, whereas the backward search on (f_2, f_1) simultaneously updates $g_1^{min}(s_{start})$. We also add a pruning criterion to discard nodes violating the primary upper bound $g_1^{min}(s_{start})$ in line 29. To achieve the backward search, we simply reverse the search direction and the objective ordering. For example, instead of $g_2^{min}(s_{goal})$ and $h_1(s(x))$ in Algorithm 2 we will have $g_1^{min}(s_{start})$ and $h'_2(s(x))$ respectively (the backward search establishes its f -values using \mathbf{h}'). Note that each search has an independent *Open* list. Now we describe our contributions to the individual searches of BOBA* followed by their formal presentation in Lemmas 10-12.

Early solution update. This strategy allows the search to update the secondary upper bound and establish a tentative solution before reaching the *goal* state. This is done via line 17 of Algorithm 2 by coupling nodes with their complementary shortest path to *goal*. If the joined path is valid, the corresponding node is then temporarily added to the solution set knowing that solution nodes with a state other than s_{goal} (or $s(x) \neq s_{goal}$) must be joined with their complementary shortest path. This strategy can be further improved by not expanding nodes for which we have a unique non-dominated complementary path. This heuristic is incorporated in line 22 and is formalised in Lemma 10.

Secondary heuristic tuning. Bi-directional search provides our algorithm with a great opportunity to further improve the quality of the preliminary heuristics. Since the main search of BOBA* has more information about non-dominated paths to states and constantly updates upper bounds, there can be more outliers that our preliminary heuristic searches are

■ **Algorithm 2** Enhanced forward Bi-Objective A* (BOA*_{enh}) in (f_1, f_2) objective ordering.

Inputs: A problem instance $(G, \mathbf{cost}, s_{start}, s_{goal})$ and heuristics $(\mathbf{h}, \mathbf{ub}, \mathbf{h}')$
Output: A set of cost-unique Pareto-optimal solutions

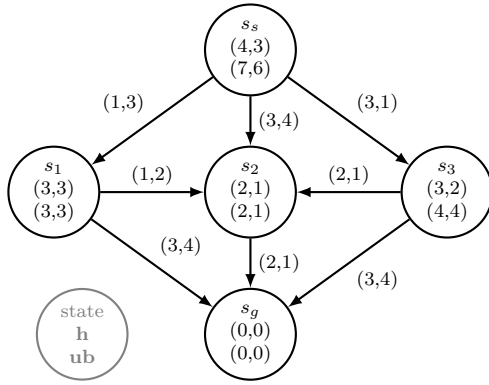
```

1  $Open \leftarrow \emptyset, Sol \leftarrow \emptyset$ 
2  $g_1^{min}(s) \leftarrow g_2^{min}(s) \leftarrow \infty$  for each  $s \in S$ 
3  $x \leftarrow$  new node with  $s(x) = s_{start}$ 
4  $\mathbf{g}(x) \leftarrow (0, 0), \mathbf{f}(x) \leftarrow (h_1(s_{start}), h_2(s_{start})), \mathbf{parent}(x) \leftarrow Null$ 
5 Add  $x$  to  $Open$ 
6 while  $Open \neq \emptyset$  do
7   Remove a node  $x$  with the lexicographically smallest  $(f_1, f_2)$  values from  $Open$ 
8*  if  $f_1(x) \geq g_1^{min}(s_{start})$  then break
9   if  $g_2(x) \geq g_2^{min}(s(x))$  or  $f_2(x) \geq g_2^{min}(s_{goal})$  then continue
10* if  $g_2^{min}(s(x)) = \infty$  then  $h'_1(s(x)) \leftarrow g_1(x)$ 
11    $g_2^{min}(s(x)) \leftarrow g_2(x)$ 
12   if  $s(x) = s_{goal}$  then
13*      $z \leftarrow$  last node in  $Sol$ 
14*     if  $(z \neq Null \text{ and } f_1(z) = f_1(x))$  then Remove  $z$  from  $Sol$ 
15     Add  $x$  to  $Sol$ 
16     continue
17*  if  $g_2(x) + ub_2(s(x)) < g_2^{min}(s_{goal})$  then
18*     $g_2^{min}(s_{goal}) \leftarrow g_2(x) + ub_2(s(x))$ 
19*     $z \leftarrow$  last node in  $Sol$ 
20*    if  $(z \neq Null \text{ and } f_1(z) = f_1(x))$  then Remove  $z$  from  $Sol$ 
21*    Add  $x$  to  $Sol$ 
22*    if  $h_1(s(x)) = ub_1(s(x))$  then continue
23  for all  $t \in Succ(s(x))$  do
24     $y \leftarrow$  new node with  $s(y) = t$ 
25     $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}(s(x), t)$ 
26     $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$ 
27     $\mathbf{parent}(y) \leftarrow x$ 
28    if  $g_2(y) \geq g_2^{min}(t)$  or  $f_2(y) \geq g_2^{min}(s_{goal})$  then continue
29*    if  $f_1 \geq g_1^{min}(s_{start})$  then continue
30    Add  $y$  to  $Open$ 
31 return  $Sol$ 

```

not aware of. Therefore, benefiting from the main property of BOA* (finding non-dominated nodes in order), we can tune our findings over the preliminary searches and empower the pruning by bound strategy of the concurrent search in the opposite direction. This tuning is done in $O(1)$ time by updating the secondary heuristics of the reverse direction via line 10 of Algorithm 2. Note that h'_1 denotes the secondary heuristic in the backward search where BOBA* uses f_2 as its primary cost. We discuss the correctness of this technique in Lemma 12.

► **Example 9.** We explain these strategies by just running the forward search of BOBA* for the graph in Figure 2 and iterations in Table 2. In the first iteration, the forward search explores the node associated with the start state s_s . Since the primary (heuristic) cost-optimum path from s_s is initially valid, the search immediately updates its secondary upper bound via the early solution update strategy by setting $g_2^{min}(s_g) = 6$ and adds the node into the Sol set with costs $(4, 6)$. During the s_s expansion, we notice that the extended path for state s_1 is invalid ($f_2(y) \geq g_2^{min}(s_g)$ or $3 + 3 \geq 6$). Therefore, the partial path to s_1 is pruned meaning that s_2 is no longer reachable via its primary cost optimum path. Nodes generated for states s_2 and s_3 , however, are successfully added to $Open$. In the second iteration, the algorithm picks the node associated with s_2 (with higher priority). Now, since this is the first time we see s_2 being expanded, and since future visits will always have higher costs (via s_3 with $g_1 = 5$ for example), we can update the lower bound of reaching



It.	Open list ($s(x)$, $\mathbf{g}(x)$, $\mathbf{f}(x)$)	Sol found	Update $g_2^{min}(s_g)$
1	$\uparrow(s_s, (0,0), (4,3))$	(4,6)	$\infty \rightarrow 6$
2	$\uparrow(s_2, (3,4), (5,5))$ $(s_3, (3,1), (6,3))$	(5,5)	$6 \rightarrow 5$
3	$\uparrow(s_3, (3,1), (6,3))$		
4	$\uparrow(s_2, (5,2), (7,3))$	(7,3)	$5 \rightarrow 3$
5	empty		

parent arrays	s_s	s_1	s_2	s_3	s_g
par_state	[Null]		$[s_s, s_3]$	$[s_s]$	
par_path_id	[Null]		[1, 1]	[1]	

■ **Figure 2** Left: An example graph with **cost** on the edges, and with (state, **h**, **ub**) inside the nodes. Right: Status of the *Open* list, new solution (*Sol*) and secondary upper bound $g_2^{min}(s_g)$ in every iteration (It.) for the forward search on the (f_1, f_2) ordering. Symbol \uparrow beside nodes denotes the expanded min-cost node. The second table shows the status of the parent arrays of the states when the search terminates.

s_2 from s_s knowing that all possible shorter paths have already been invalidated. This is done by updating $h'_1(s_2) = 3$. Note that from the preliminary heuristics, we already had $h'_1(s_2) = 2$ (lower bound from s_s to s_2). After this update, the backward search would have better quality secondary lower bounds and can effectively prune more nodes (the backward primary heuristic is h'_2). We skip the backward search for now and continue with our forward expansions. As coupling the node (associated with s_2) with its (complementary) primary cost-optimum path yields a valid complete path, the search updates its secondary upper bound and temporarily adds the node to the *Sol* set with costs (5, 5). The search also skips expanding s_2 as it does not offer any non-dominated path to s_g . In the third iteration, the node associated with state s_3 is picked. This time, s_3 is expanded since coupling does not yield valid path. During the s_3 expansion, the search finds s_g invalid but adds s_2 into *Open*. In the fourth iteration, the node associated with s_2 is the only node in *Open* which reveals the final solution with costs (7, 3), again with the early solution update strategy. This last solution also verifies that the temporary solution found in the second iteration is now a valid non-dominated solution, since the primary cost of the last solution is larger than that of the second solution ($5 < 7$).

Now we formally prove the correctness of the presented techniques as follows.

► **Lemma 10.** *At every iteration, if $g_2(x) + ub_2(s(x)) < g_2^{min}(s_{goal})$, the next solution has a primary cost of $f_1(x)$ and a secondary cost of at most $g_2(x) + ub_2(s(x))$. Node x is also a terminal node if $h_1(s(x)) = ub_1(s(x))$.*

Proof. If the joined path is valid (its secondary cost is within the bounds), expanding nodes on the complementary shortest path will definitely navigate us to s_{goal} with a valid secondary cost as they offer the same f_1 -value. This means we can efficiently update the secondary upper bound earlier assuming that a potential solution path is already established. Therefore, valid joined paths determine the primary cost f_1 of the next solution along with setting a new upper bound for the secondary cost f_2 . Furthermore, given the secondary cost as a tie-breaker in the preliminary heuristic searches, states with $h_1(s(x)) = ub_1(s(x))$ would only offer one complementary path optimum for both objectives. As none of the states on the

complementary path would offer an alternative non-dominated path to s_{goal} , the search can save time by not expanding such terminal nodes. Therefore, nodes with $h_1(s(x)) = ub_1(s(x))$ are terminal nodes if they appear on any solution path. ◀

The early solution update strategy above guarantees that the primary cost f_1 of the next solution is determined by the joined path, but this does not apply to its secondary cost. E.g., we might see two consecutive temporary solutions with the same f_1 -value but different (sequentially) valid secondary costs. Therefore, the search needs to make sure that the previously added solution is not dominated by the next potential solution, and if it is dominated, it must be removed from the non-dominated solution set Sol . We address this matter in $O(1)$ time by checking our last (temporary) solution against new solutions for dominance as shown in lines 13-14 and 19-20 of Algorithm 2. This pruning is formally stated in the following Lemma 11.

► **Lemma 11.** *Given z and x as the last and new temporary solution nodes respectively, node z represents a non-dominated solution if $f_1(z) < f_1(x)$. The temporary solution node z is dominated by x if $f_1(z) = f_1(x)$.*

Proof. Since the secondary cost of the new solution x is already checked to be smaller than that of the last solution z stored in $g_2^{min}(s_{goal})$, we have $f_2(x) < g_2^{min}(s_{goal})$ if $s(x) = s_{goal}$ or $g_2(x) + ub_2(s(x)) < g_2^{min}(s_{goal})$ if $s(x) \neq s_{goal}$. On the other hand, since x is the new potential solution and the search explores nodes in an increasing order of f_1 -values, we must have $f_1(z) \leq f_1(x)$. Therefore, if $f_1(z) < f_1(x)$, we can see that the temporary solution z is now a non-dominated solution. Otherwise, if $f_1(z) = f_1(x)$, the last solution z is dominated by the new solution x because the new solution offers a lower secondary cost. ◀

We now show the correctness of the heuristic tuning approach in BOBA*.

► **Lemma 12.** *The secondary heuristic tuning maintains the correctness of A^* heuristics.*

Proof. BOBA* expands partial paths in the increasing order of f -values. This means the first expanded node of every state is guaranteed to have the minimum valid primary cost g_1 in each search direction, and all of the following valid nodes will have a larger primary cost. Moreover, since BOBA* uses different objective ordering for its searches, updated lower bounds in one direction represent the secondary heuristics of the other direction. Therefore, we can guarantee that the updated secondary heuristic is still admissible as there will not be any min-cost path to states better than what their first expanded node presents. Furthermore, the tuning strategy only updates the secondary heuristics of the opposite search, i.e., $h'_1(s(x))$ in the forward and $h_2(s(x))$ in the backward search. Therefore, the preliminary primary heuristics $h_1(s(x))$ and $h'_2(s(x))$ are unchanged and the A^* searches are correct. ◀

Considering the correctness of the enhancements presented above, we now show the correctness of our BOBA* algorithm.

► **Theorem 13.** *BOBA* returns a set of cost-unique non-dominated solution paths.*

Proof. BOBA* executes two enhanced BOA* searches concurrently, each capable of finding all of the solutions. Therefore, we just need to show the correctness of the stopping criteria. Each (enhanced) BOA* searches the primary objective's domain in the increasing order of f -values and continually shrinks the secondary objective's domain every time a valid solution is found. Furthermore, since BOBA* shares the upper bounds between its searches, each search can terminate with the first node violating the main objective's upper bound (and

consequently other unexplored nodes with larger f -values in *Open*) knowing that the rest of the objective's domain has already been investigated by the concurrent search (see Figure 1). Therefore, the aggregation of the solutions found in each search yields a complete set of cost-unique non-dominated solutions. ◀

4 Practical Considerations

As BOA* enumerates all non-dominated paths, the size of *Open* can grow exponentially over the course of search. Furthermore, the huge number of nodes in difficult cases may result in major memory issues. For instance, for one particular case in our experiments BOA* generates two billion nodes. We now present two techniques to handle search nodes more efficiently.

More efficient *Open* list. To achieve faster operations in our *Open* lists, since the lower and upper bounds on the f -values of the nodes in BOBA* are known prior to its main searches, we replace the conventional heap-based lists with fixed-size bucket lists without tie-breaking [3]. In contrast to other problems where the bucket list is regularly resized and the list is sparsely populated, for the significant number of (cost-bounded) nodes in our problem we expect to see almost all of the buckets filled. Note that the search may also expand dominated nodes if they are not extracted in a lexicographical order (i.e., nodes are sorted based on their primary cost only), but BOBA* can still obtain cost-unique solutions via the dominance checks incorporated in lines 13-14 and 19-20 of Algorithm 2 as formally stated in the following Lemma 14.

► **Lemma 14.** *BOBA* is able to obtain cost-unique solutions even without tie-breaking in its *Open* lists.*

Proof. Let z and x be two solution nodes where $f_1(z) = f_1(x)$ and z is dominated by x . Without any tie-breaking, the search may temporarily add dominated node z to the solution set first. In the next iterations, when x is extracted, the search performs a quick dominance check by comparing the f_1 -value of the new node x against that of the previous solution z and substitutes the dominated solution with the new solution x if $f_1(z) = f_1(x)$, as already shown in the early solution update strategy and Lemma 11 in detail. Therefore, BOBA* computes cost-unique non-dominated solutions even without tie-breaking. ◀

Memory efficient backtracking. Creating nodes is necessary to appropriately navigate the search to valid solution paths. Each new node occupies a constant amount of memory and conventionally contains essential information about paths such as costs and also back-pointers for solution path construction. Considering the difficulty of the problem and the significant number of generated nodes, we suggest a more memory efficient approach for the solution path construction in BOBA*. Since BOBA* only expands nodes once, we propose to recycle the memory used to store heavy processed nodes, while storing their backtracking information in other compact data structures. This technique results in a major reduction in memory use as part of the nodes' information would no longer be required for backtracking. We explain our compact approach using an example from Figure 2. Assume that in the second iteration of the algorithm, we want to store the backtracking information of the node corresponding to s_2 with s_s as the parent state. To this end, we keep two (initially empty) dynamic arrays for each state: one to store the parent state of the node `par_state`, and another to look up the corresponding path index in the parent state `par_path_id`. For our example, since the first

path to s_2 is derived from the first non-dominated path of s_s , we store this sequence in s_2 as `par_state[1]= s_s` and `par_path_id[1]=1`. Similarly, for the second expansion of s_2 with s_3 as the parent in the fourth iteration, we update s_2 arrays, this time with `par_state[2]= s_3` and `par_path_id[2]=1`. Figure 2 also shows the situation of our parent arrays when the forward search terminates. As a further optimisation, we can store the index of incoming edges (which are usually very small integers) instead of parent states. We will investigate the impacts of this compression on memory usage in the following section.

5 Empirical Study and Analysis

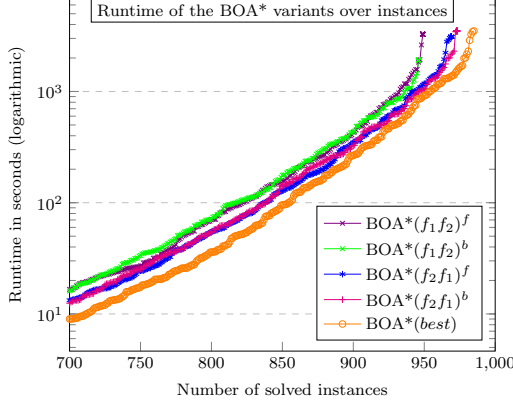
We compare our BOBA* with recent algorithms designed for the bi-objective search problem. The selected algorithms are the bi-objective variants of Dijkstra’s algorithm (Dij) and bi-directional Dijkstra (Bi-Dij) from [12], and bi-objective A* (BOA*) from [17]. We use all seven benchmark instances of [12] which include 700 random *start-goal* pairs from the large road networks in the 9th DIMACS challenge [4] with (*distance*, *time*) as objectives. To further challenge the algorithms, we used the competition’s random pair generator to design an additional set of 300 test-cases for the larger networks in the DIMACS instances: E, W and CTR (100 cases each) with up to 15 M nodes and 33 M edges. The details of the instances can be found in [4].

Implementation. We implemented our BOBA* algorithms based on a parallel framework using two cores in C++ and used the C implementations of the Dij, Bi-Dij and BOA* algorithms kindly provided to us by their authors. We also fixed the scalability and tie-breaking issues in the standard BOA* algorithm before running the experiments. All code was compiled with O3 optimisation settings using the GCC7.4 compiler. Our codes are publicly available². We ran the 1,000 experiments on an Intel Xeon E5-2660V3 processor running at 2.6 GHz and with 128 GB of RAM, under the SUSE Linux 12.4 environment and with a one-hour timeout.

BOA* analysis. The search in BOA* can be performed in different directions and objective orders, resulting in four variants. We also consider the virtual best version of the four, called BOA*_{best} (essentially assuming an oracle that could select the best variant). Figure 3 is a cactus plot comparing the performance of all BOA* variants including the virtual best, showing how many instances can be solved in a given time (the plot only shows the longest running 300 instances). Backward BOA* in the (f_1, f_2) order (in green) is the weakest variant, but the other variants perform quite similarly, and it is difficult to declare a clear winner. The performance of the virtual variant BOA*_{best} shows that an ideally-tuned BOA* can be up to two times better than its standard version on average, but is still unable to solve 15 cases to optimality within the time limit (see Table 2).

Memory. We investigate the impact of our compact approach for the solution paths construction in BOBA*. Table 1 compares the memory usage of our proposed compact approach against the conventional backtracking approach on part of the benchmark instances. In order to measure the overall space requirement of the main search, we ignore the memory required for graph construction, shared libraries and heuristics, allocated prior to the search. The results show that BOBA* can solve all of the instances with both approaches within

² <https://bitbucket.org/s-ahmadi>



■ **Figure 3** Performance of the BOA* variants.

■ **Table 1** BOBA* memory usage for the conventional and compact backtracking approaches.

Inst.	Saving Approach	Mem. (MB)	
		Avg.	Max
NE	Conv.	307	7618
	Compact	61	1186
CAL	Conv.	326	5421
	Compact	62	1009
LKS	Conv.	5585	54331
	Compact	955	9411
E	Conv.	5836	62963
	Compact	999	10895
W	Conv.	5877	79602
	Compact	925	10498
CTR	Conv.	15835	99108
	Compact	2662	21749

the time limit, but the compact approach runs slightly faster and is five times more efficient on average in terms of memory. For the most difficult case in the experiments, the required memory of the compact approach can be as low as 21 GB (allocating 15M nodes with recycling) where the conventional approach needs 96 GB (allocating 1B nodes). Note that both approaches nearly expand the same number of nodes to solve the cases to optimality.

BOBA* performance. We compare the performance of our parallel BOBA* algorithm with the state-of-the-art Dij, Bi-Dij and BOA* algorithms from the literature. Table 2 shows the summary of experimental results for the 100 cases of each instance. For unsolved cases, we generously assume a runtime of one hour (the timeout). We also report the average memory usage of the main search of each algorithm over solved cases, ignoring the space allocated for their initialisation phase. The results in Table 2 show that the standard BOA* algorithm runs faster, needs less memory compared to both Dij and Bi-Dij algorithms and solves more instances. However, our new BOBA* outperforms BOA* in all of the instances, showing an (arithmetic) average speed up of 16 over all of the individual cases. For the average runtime of all instances, BOBA* is around five times faster than BOA*. We also compare the algorithms' performance over the solved instances for both CPU time and memory usage in Figure 4. As shown for both metrics, BOBA* delivers superior performance to its competitors by solving all of the instances to optimality within the time limit and with a maximum memory usage of 21 GB, compared to the nearly full (128 GB) memory usage of other algorithms in difficult instances. BOBA* also shows a massive speed up in the easy cases due to its efficient initialisation phase. It can solve 282 cases before BOA* solves its easiest case. Moreover, the figure shows that BOBA* completes the task eight times more efficiently in terms of memory than BOA* on average. Note that because of the difficulties in reporting the memory usage, we allow 1 MB tolerance in our experiments.

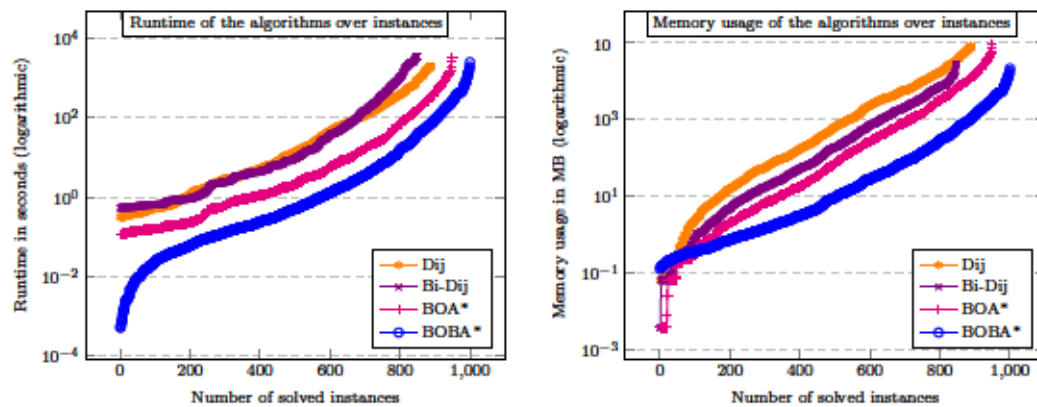
Multi-threading. We investigate the impact of multi-threading in BOBA* by running the (unmodified) algorithm on a single core instead of two cores, allowing decisions on scheduling of the threads to be made by the operating system. We compare single-core BOBA*_{1c} with the virtual best variant BOA*_{best} and our BOBA* with two cores in Table 2. The results show a slowdown of around 1.8 compared to parallel BOBA*, but it still outperforms

■ **Table 2** Number of solved cases ($|S|$), runtime (in seconds) and average memory usage (Mem.) of algorithms over instances (Ins.). Memory in MB for the main search over solved cases.

Alg.	Ins.	$ S $	Runtime (s)			Mem.	Ins.	$ S $	Runtime (s)			Mem.
			Min	Avg.	Max				Min	Avg.	Max	
BiDij	NY	100	0.53	0.92	6.66	21	CAL	98	4.04	168.58	3600.00	1206
Dij		100	0.31	1.47	16.98	75		100	2.73	88.93	1105.57	4283
BOA*		100	0.11	0.22	1.70	9		100	0.89	24.50	538.40	893
BOA* _{best}		100	0.11	0.16	0.65	6		100	0.89	8.63	187.04	498
BOA* _{enh}		100	0.01	0.12	0.67	2		100	0.02	6.69	147.44	67
BOBA* _{1c}		100	0.01	0.11	0.59	7		100	0.01	6.08	116.10	97
BOBA*		100	0.00	0.08	0.40	2		100	0.00	3.75	64.80	62
BiDij	BAY	100	0.61	1.32	11.96	35	LKS	69	6.12	1610.14	3600.00	2597
Dij		100	0.36	2.01	19.71	107		81	4.13	936.23	3600.00	11394
BOA*		100	0.13	0.38	4.10	19		89	1.30	528.12	3600.00	4854
BOA* _{best}		100	0.13	0.23	1.26	13		100	1.28	224.42	3500.80	9374
BOA* _{enh}		100	0.01	0.19	1.20	3		100	0.02	97.05	1077.96	787
BOBA* _{1c}		100	0.01	0.19	1.08	10		100	0.02	129.64	1488.41	1123
BOBA*		100	0.00	0.13	0.86	2		100	0.00	69.68	812.17	955
BiDij	COL	100	0.84	6.84	147.55	118	E	64	8.08	1611.10	3600.00	2223
Dij		100	0.52	6.27	111.81	348		82	5.48	1034.61	3600.00	16156
BOA*		100	0.19	1.20	20.53	77		89	1.72	552.64	3600.00	5299
BOA* _{best}		100	0.18	0.58	7.03	49		98	1.72	293.27	3600.00	8701
BOA* _{enh}		100	0.01	0.54	10.46	7		100	0.02	110.69	1684.94	850
BOBA* _{1c}		100	0.02	0.42	5.50	21		100	0.02	143.08	1818.63	1160
BOBA*		100	0.00	0.34	6.58	6		100	0.00	75.94	952.32	999
BiDij	FLA	100	2.11	51.49	1088.49	808	W	69	14.38	1585.11	3600.00	3476
Dij		100	1.37	52.34	1048.67	2630		74	10.04	1220.44	3600.00	12722
BOA*		100	0.48	6.42	153.07	276		94	3.14	416.94	3600.00	7705
BOA* _{best}		100	0.48	3.22	36.32	202		98	3.14	253.85	3600.00	8043
BOA* _{enh}		100	0.01	2.05	34.47	22		100	0.04	93.16	1792.57	784
BOBA* _{1c}		100	0.01	2.06	27.98	43		100	0.04	130.81	1834.67	1134
BOBA*		100	0.00	1.31	19.86	25		100	0.02	70.41	971.67	925
BiDij	NE	99	3.31	181.67	3600.00	1367	CTR	48	40.41	2666.66	3600.00	4904
Dij		100	2.18	68.41	1306.04	3281		51	29.29	2163.50	3600.00	16149
BOA*		100	0.73	16.83	332.36	587		77	8.46	1124.03	3600.00	9828
BOA* _{best}		100	0.70	10.51	332.01	533		89	8.46	745.16	3600.00	12418
BOA* _{enh}		100	0.02	4.79	97.25	49		100	0.03	340.50	2953.12	2178
BOBA* _{1c}		100	0.02	5.71	154.51	82		98	0.03	461.07	3600.00	2644
BOBA*		100	0.00	3.41	90.01	61		100	0.02	246.01	2496.95	2662

BOA*_{best}, solving more instances and showing an (arithmetic) average speed-up of six over all of the individual cases. Note that this virtual best version BOA*_{best} does not exist, and the results are based on the best timings obtained via four individual runs of the standard BOA* algorithm.

Enhanced BOA*. To measure the contributions of our improvements to the uni-directional bi-objective search, we analyse the performance of the enhanced variant BOA*_{enh} with the speed-up techniques above. This variant is obtained by switching off the backward search of BOBA*. Based on the results given in Table 2, BOA*_{enh} outperforms BOA*_{best} in almost all of the cases and shows a comparable performance to BOBA*_{1c}, solving a few more cases in the CTR map and using less memory on average. Comparing the maximum runtime over instances, we can see that BOBA*_{1c} is faster than BOA*_{enh} in half of the instances (maps NY, BAY, COL, FLA and CAL). Nonetheless, given the results in Table 2, BOBA* is still superior to BOA*_{enh} showing a speed-up factor of 1.5 on average.



■ **Figure 4** Cactus plots of algorithms' performance. Left: Runtime. Right: Search memory usage.

Bucket vs. heap. We found BOBA* with the bucket-based *Open* list around 1.8 times faster than BOBA* with heap for the same set of instances on average. Nonetheless, BOBA* with heap is still 2.2 times faster than standard heap-based BOA* (average over instances).

6 Conclusion

This paper introduced BOBA*, a bi-directional version of the state-of-the-art BOA* algorithm for bi-objective search. Our new algorithm explores the graph from both (forward and backward) directions in different objective orders in parallel. We enrich BOBA* with more efficient approaches for both the initial heuristic procedure and the solution path construction. We also present several speed up strategies to enhance BOBA*'s searches in various scenarios. Our experiments show that BOBA* outperforms the state-of-the-art algorithms in both runtime and memory use, solving all of the 1,000 benchmark cases to optimality in one hour timeout. Furthermore, compared to BOA*, BOBA* is five times faster and needs eight times less memory on average. Additional experiments reveal that the single-core version of BOBA* is around 1.8 times slower than the parallel version but still superior to the virtual best variant of BOA* and shows a comparable performance to BOA* enhanced with the speed-up strategies of this study.

References

- 1 W. Matthew Carlyle and R. Kevin Wood. Near-shortest and k-shortest simple paths. *Networks*, 46(2):98–109, 2005. doi:10.1002/net.20077.
- 2 Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.
- 3 Eric V. Denardo and Bennett L. Fox. Shortest-route methods: 1. reaching, pruning, and buckets. *Oper. Res.*, 27(1):161–186, 1979. doi:10.1287/opre.27.1.161.
- 4 DIMACS. 9th dimacs implementation challenge - shortest paths, 2005. URL: <http://users.diag.uniroma1.it/challenge9>.
- 5 Daniel Duque, Leonardo Lozano, and Andrés L. Medaglia. An exact method for the biobjective shortest path problem for large-scale road networks. *Eur. J. Oper. Res.*, 242(3):788–797, 2015. doi:10.1016/j.ejor.2014.11.003.
- 6 F Guerriero and R Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of optimization theory and applications*, 111(3):589–613, 2001.

- 7 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.
- 8 Enrique Machuca and Lawrence Mandow. Multiobjective heuristic search in road maps. *Expert Syst. Appl.*, 39(7):6435–6445, 2012. doi:10.1016/j.eswa.2011.12.022.
- 9 Francisco Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64:60–70, 2015. doi:10.1016/j.cor.2015.05.007.
- 10 Andrea Raith. Speed-up of labelling algorithms for biobjective shortest path problems. In *Proceedings of the 45th annual conference of the ORSNZ , 29-30 Nov 2010, Auckland, New Zealand*, page 313–322. Operations Research Society of New Zealand, 2010. URL: <http://hdl.handle.net/2292/9789>.
- 11 Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Comput. Oper. Res.*, 36(4):1299–1331, 2009. doi:10.1016/j.cor.2008.02.002.
- 12 Antonio Sedeño-Noda and Marcos Colebrook. A biobjective dijkstra algorithm. *Eur. J. Oper. Res.*, 276(1):106–118, 2019. doi:10.1016/j.ejor.2019.01.007.
- 13 Liang Shen, Hu Shao, Ting Wu, William HK Lam, and Emily C Zhu. An energy-efficient reliable path finding algorithm for stochastic road networks with electric vehicles. *Transportation Research Part C: Emerging Technologies*, 102:450–473, 2019.
- 14 Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, Solomon Eyal Shimony, and Avi Hayoun. Enriching non-parametric bidirectional search algorithms. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 2379–2386. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33012379.
- 15 Anders J. V. Skriver and Kim Allan Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Comput. Oper. Res.*, 27(6):507–524, 2000. doi:10.1016/S0305-0548(99)00037-4.
- 16 Nathan R. Sturtevant and Ariel Felner. A brief history and recent achievements in bidirectional search. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 8000–8007. AAAI Press, 2018. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17232>.
- 17 Carlos Hernández Ulloa, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In J. Christopher Beck, Olivier Buffet, Jörg Hoffmann, Erez Karpas, and Shirin Sohrabi, editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 143–151. AAAI Press, 2020. URL: <https://aaai.org/ojs/index.php/ICAPS/article/view/6655>.
- 18 Aphrodite Veneti, Angelos Makrygiorgos, Charalampos Konstantopoulos, Grammati E. Pantziou, and Ioannis A. Vetsikas. Minimizing the fuel consumption and the risk in maritime transportation: A bi-objective weather routing approach. *Comput. Oper. Res.*, 88:220–236, 2017. doi:10.1016/j.cor.2017.07.010.

A Unified Approach for All Pairs Approximate Shortest Paths in Weighted Undirected Graphs

Maor Akav ✉

Bar-Ilan University, Ramat Gan, Israel

Liam Roditty ✉

Bar-Ilan University, Ramat Gan, Israel

Abstract

Let $G = (V, E)$ be a weighted undirected graph with n vertices and m edges, and let $d_G(u, v)$ be the length of the shortest path between u and v in G . In this paper we present a unified approach for obtaining algorithms for all pairs approximate shortest paths in weighted undirected graphs. For every integer $k \geq 2$ we show that there is an $\tilde{O}(n^2 + kn^{2-3/k}m^{2/k})$ expected running time algorithm that computes a matrix M such that for every $u, v \in V$:

$$d_G(u, v) \leq M[u, v] \leq (2 + \frac{k-2}{k})d_G(u, v).$$

Previous algorithms obtained only specific approximation factors. Baswana and Kavitha [FOCS 2006, SICOMP 2010] presented a 2-approximation algorithm with expected running time of $\tilde{O}(n^2 + m\sqrt{n})$ and a $7/3$ -approximation algorithm with expected running time of $\tilde{O}(n^2 + m^{2/3}n)$. Their results improved upon the results of Cohen and Zwick [SODA 1997, JoA 2001] for graphs with $m = o(n^2)$. Kavitha [FSTTCS 2007, Algorithmica 2012] presented a $5/2$ -approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$.

For $k = 2$ and $k = 3$ our result gives the algorithms of Baswana and Kavitha. For $k = 4$, we get a $5/2$ -approximation algorithm with $\tilde{O}(n^{5/4}m^{1/2})$ expected running time. This improves upon the running time of $\tilde{O}(n^{9/4})$ due to Kavitha, when $m = o(n^2)$.

Our unified approach reveals that all previous algorithms are a part of a family of algorithms that exhibit a smooth tradeoff between approximation of 2 and 3, and are not sporadic unrelated results. Moreover, our new algorithm uses, among other ideas, the celebrated approximate distance oracles of Thorup and Zwick [STOC 2001, JACM 2005] in a non standard way, which we believe is of independent interest, due to their extensive usage in a variety of applications.

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

Keywords and phrases Graph algorithms, Approximate All Pairs of Shortest Paths, Distance Oracles

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.4

1 Introduction

Computing All Pairs of Shortest Paths (APSP) is one of the most fundamental problems in Computer Science. The fastest known algorithms for APSP in weighted graphs run in $\min\{\tilde{O}(mn), n^3 / \exp(\sqrt{\log n})\}$ [27, 19, 20].

In unweighted undirected graphs the fastest known APSP algorithms run in $\tilde{O}(\min\{mn, n^\omega\})$ time¹ [21], where $\omega < 2.373$ is the exponent of square matrix multiplication [28, 17, 24], n is the number of vertices and m is the number of edges. For an extension of this result to undirected graphs with integral weights see [22] and to directed graphs see [30]. Fast Matrix Multiplication (FMM) algorithms hide large constants and are thus far from being practical. A fundamental research question is whether one can obtain fast “combinatorial” algorithms, that can be implemented.

¹ \tilde{O} notation hides polylogarithmic factors



Aingworth, Chekuri, Indyk and Motwani [1] initiated the research on efficient APSP algorithms in unweighted undirected graphs, that settle for an approximated solution and do not use FMM. An all pairs approximate shortest paths (APASP) algorithm has (α, β) -approximation if every distance X is estimated by the algorithm to be at least X and at most $\alpha X + \beta$, where α is the multiplicative approximation and β is the additive approximation. Aingworth et al. [1] presented a $(1, 2)$ -approximation that runs in $\tilde{O}(n^{2.5})$.

Dor, Halperin and Zwick [10] improved and generalized the results of [1]. For every even $k > 2$, they presented a generalized scheme with $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ running time and an additive approximation of k . For $k = 2$, the running time is $\tilde{O}(\min\{n^{\frac{3}{2}}m^{\frac{1}{2}}, n^{\frac{7}{3}}\})$ and the additive approximation is 2. They also obtained an algorithm with $\tilde{O}(n^2)$ running time and $(3, 0)$ -approximation. Berman and Kasiviswanathan [6] improved this result and obtained an algorithm with the same running time of $\tilde{O}(n^2)$ and $(2, 1)$ -approximation. The question of obtaining efficient APASP algorithms for *unweighted* undirected graphs was considered by many subsequent works [11, 5, 3, 23, 16, 2].

Cohen and Zwick [9] extended the results of Dor et al. [10] to *weighted* undirected graphs. They obtained an $\tilde{O}(n^2)$ time algorithm with a $(3, 0)$ -approximation, an $\tilde{O}(n^{\frac{3}{2}}m^{\frac{1}{2}})$ time algorithm with a $(2, 0)$ -approximation and also $\tilde{O}(n^{\frac{7}{3}})$ time algorithm with $(7/3, 0)$ -approximation. Their analysis of the $(3, 0)$ -approximation algorithm allowed them to obtain a general scheme with a running time of $\tilde{O}(n^{2-1/k}m^{1/k})$ that for every pair $u, v \in V$ computes an additive approximation of $2 \sum_{i=1}^{k-1} w_i$, where w_i is the i th heaviest edge on the shortest path between u and v . (For a similar result see also [12].) This is still a $(3, 0)$ -approximation in the worst case.

Baswana and Kavitha [4] presented several algorithms that perform better than the algorithms of Cohen and Zwick when $m = o(n^2)$. They presented a $(2, 0)$ -approximation algorithm with expected running time of $\tilde{O}(n^2 + m\sqrt{n})$ and a $(7/3, 0)$ -approximation algorithm with expected running time of $\tilde{O}(n^2 + m^{2/3}n)$. Notice that both algorithms have the same running times as those of Cohen and Zwick when $m = \Theta(n^2)$. Kavitha [15] presented a $(5/2, 0)$ -approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$. Baswana and Kavitha [4] presented also a $(2, W)$ -approximation algorithm, where W is the largest edge weight, with expected running time of $\tilde{O}(\min\{n^2, m\sqrt{n}\})$. This is still a $(3, 0)$ -approximation in the worst case. A deterministic algorithm with the same bounds was presented by Berman and Kasiviswanathan [6].

It stems from all the previous results mentioned above that for $m = \Theta(n^2)$ there is a $(2 + (k - 2)/k, 0)$ -approximation algorithm with an expected running time of $\tilde{O}(n^{2+1/k})$, for every $k \in \{2, 3, 4\}$. For $k \in \{2, 3\}$, there is a $(2 + (k - 2)/k, 0)$ -approximation algorithm with expected running time of $\tilde{O}(n^2 + n^{2-\frac{2}{k}}m^{\frac{2}{k}})$. Therefore, in light of the existing results and the lack of progress in improving them for more than 15 years, a natural research question that arises is, whether there is a general scheme of algorithms with $(2 + (k - 2)/k, 0)$ -approximation and $\tilde{O}(n^{2+1/k})$ running time, for every $k > 4$ or even more generally, is there a general scheme of algorithms with $(2 + (k - 2)/k, 0)$ -approximation that work better for sparse graphs and have a running time of $\tilde{O}(n^2 + n^{2-\frac{3}{k}}m^{\frac{2}{k}})$, for every $k > 3$. Obtaining the latter scheme requires first to improve the $(5/2, 0)$ -approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$ of Kavitha [15]. In this paper we answer the more general question positively and prove:

► **Theorem 1.** *For every integer $k \geq 2$, there is an APASP algorithm with expected running time of $\tilde{O}(n^2 + m^{\frac{2}{k}}n^{2-\frac{3}{k}})$ and multiplicative approximation of $2 + \frac{k-2}{k}$.*

For $k = 2, 3$ we get the results of Baswana and Kavitha [4]. For $k = 4$, we improve upon the result of Kavitha [15] and get a $(5/2, 0)$ -approximation algorithm with $\tilde{O}(n^2 + n^{\frac{5}{4}}m^{\frac{1}{2}})$ expected running time, which is better than $\tilde{O}(n^{9/4})$ when $m = o(n^2)$.

The main contribution of Theorem 1 is in unifying all previous algorithms into one general systematic scheme which reveals for the previous known algorithms with 2, $7/3$ and $5/2$ approximation, that rather than being accidental ad-hoc results, they are part of a family of algorithms that exhibits a smooth tradeoff between time and approximation.

Another aspect of Theorem 1 is that the current state of the art in conditional lower bounds only rules out the existence of a $(2 - \varepsilon)$ -approximation algorithm whose running time is better than the running time of FMM. No conditional lower bounds are currently known for approximation factors in the range between 2 and 3. Our result shed a light on this range from the upper bound perspective.

From the technical perspective our result is obtained by a combination of several observations with one important key ingredient that might be useful for generalizing also other algorithms that are based on distance computation. Very roughly speaking, the algorithms for $k = 2, 3$ of [4] and for $k = 4$ of [15] are obtained by using a case analysis in which one of the cases uses the following approach. For every $u, v \in V$ there is some edge $(a, b) \in E$ on a shortest path between u and v that is used to get an estimation to the distance between u and v based on approximate distance queries between u and b and v and a . We develop a generalization of this idea which is based on the query procedure of the *approximate distance oracles* (ADO) of Thorup and Zwick [26]. However, this key ingredient in its own is not enough to obtain Theorem 1. A tighter analysis of the algorithm of [4] for the case that $k = 2$ and a careful combination of it with our generalization technique is required.

The ADO of Thorup and Zwick [26] plays a pivotal role in many results related to approximating distances. Formally, Thorup and Zwick showed that for any integer $k \geq 1$ it is possible to preprocess a weighted undirected graph in $O(kmn^{1/k})$ expected time and to create ADO of size $O(kn^{1+1/k})$. For every $u, v \in V$ a query returns in $O(k)$ time a $(2k - 1, 0)$ -approximation. Many of the subsequent works on ADO were focused on improving the preprocessing time. (For more details see for example [4, 29, 23].) Baswana and Kavitha [4] showed that for $k > 2$ it is possible to compute ADO in $\tilde{O}(\min\{n^2, kmn^{1/k}\})$ expected running time. Other aspects of ADO were considered as well. For more details see for example [18, 29, 7, 8, 13, 14].

The rest of this paper is organized as follows. In the next section we present notations and review previous works. In Section 3 we present the main ingredients needed for obtaining the generalization. In Section 4 we present a couple of additional ingredients whose combination with the generalization presented in Section 3 yields the proof of Theorem 1.

2 Preliminaries

Let $G = (V, E)$ be a weighted undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Let $\mathbf{w} : E \rightarrow \mathbb{R}^+$ be a weight function on the edges of G . For a vertex $u \in V$, let $N(u)$ be the set of neighbours of u including u itself. Let E_u be the set of incident edges of u . Let $E_u(i)$ be the i lightest edges² that are incident to u and let $N(u, i) = \{v \mid (u, v) \in E_u(i)\}$. Let $E(i) = \cup_{u \in V} E_u(i)$. Let $u, v \in V$ and let $d_G(u, v)$ be the distance between u and v in G , that is, the length of the shortest path between u and v . Let $S \subseteq V$. Let $p_S(u) = \arg \min_{w \in S} d_G(u, w)$ and let $d_G(u, S) = d_G(u, p_S(u))$. If $S = \emptyset$ then $d_G(u, S) = \infty$. Notice that $p_S(s) = s$ for every $s \in S$. Let $E_S(u) = \{(u, v) \in E_u \mid \mathbf{w}(u, v) < d_G(u, S)\}$ and let $E_S = \cup_{u \in V} E_S(u)$. Notice that in the degenerate case that $S = \emptyset$ we have $E_S = E$, since $d_G(u, S) = \infty$.

² Ties are broken arbitrarily. If i is not integral we take the $\lceil i \rceil$ lightest edges.

Several variants of the next two Lemmas were used in many of the previous papers [10, 9, 26]. For our needs we use the formulation of Baswana and Kavitha [4]:

► **Lemma 2** ([4]). *Given a weighted undirected graph $G = (V, E)$ and a set $S \subseteq V$ the following holds:*

1. *If $d_G(u, v) < d_G(u, S)$ then all the edges on a shortest path between u and v are in $G(V, E_S)$ and $d_{G(V, E_S)}(u, v) = d_G(u, v)$.*
2. *$d_{G(V, E_S \cup E_{p_S(u)})}(u, p_S(u)) = d_G(u, p_S(u))$.*

► **Lemma 3** ([4]). *If $v \in V$ is added to S with probability p then $\mathbb{E}[|E_S|]$ is $O(n/p)$.*

For every $u \in V$, the *ball* of u with respect to S is $B_S(u) = \{w \mid d_G(u, w) < d_G(u, S)\}$. An example to a ball is presented in The next definition of *overlapping* balls is used implicitly in many of the previous research on ADO and APASP. This definition was stated explicitly by Kavitha [15].

► **Definition 4** ([15]). *Balls $B_S(u)$ and $B_S(v)$ overlap if $d_G(u, S) + d_G(v, S) > d_G(u, v)$.*

The next Corollary from [15] is crucial for our new algorithms.

► **Corollary 5** ([15]). *If $P(u, v)$ is a shortest path and $B_S(u)$ and $B_S(v)$ overlap then $P(u, v)$ can be divided into a portion $P(u, a)$ in $B_S(u)$, a portion $P(v, b)$ in $B_S(v)$, and an edge (a, b) ³.*

Let $S_0 \subseteq V$, $S_{i+1} \subseteq S_i$, where $i \in [0, k]$ and $k \geq 1$. The set S_{i+1} is constructed by picking every vertex of S_i independently at random with probability $p \cdot c \log n$, for some constant c . We denote these $k + 1$ vertex sets with \mathcal{S}_k^p and call them a *regular* hierarchy if $S_0 = V$ and $S_k = \emptyset$. Later we also define *augmented* and *mixed* hierarchies. We will usually refer to vertices in S_i , for $i > 0$, as pivots. For every $u \in V$ let $p_i(u) = p_{S_i}(u)$.

Thorup and Zwick [26] introduced approximate distance oracles (ADO). Given $k \geq 1$, an ADO of size $\tilde{O}(n^{1+1/k})$ can be constructed in $\tilde{O}(mn^{1/k})$ time. A query between any $u, v \in V$ returns in $O(k)$ time a $(2k - 1)$ -multiplicative approximation of the distance. At the core of the ADO is the definition of a *bunch* $B_i(u)$ for every vertex $u \in V$ and $i \in [0, k - 1]$. For a given $i \in [0, k - 1]$ and $u \in V$ a bunch $B_i(u)$ is defined as follows: $B_i(u) = \{w \in S_i \setminus S_{i+1} \mid d_G(u, w) < d_G(u, p_{i+1}(u))\}$. Notice that if we translate the definition of bunches to the terminology of balls then for every $u \in V$ and $i \in [0, k - 1]$, $B_i(u) = S_i \cap B_{S_{i+1}}(u)$. In particular, for $i = 0$ we have that the bunch $B_0(u)$ is simply the ball $B_{S_1}(u)$. Thorup and Zwick [26] proved the following Lemma on the size of the bunches.

► **Lemma 6** ([26]). *Let $p = qc \log n$, where c is a large constant and $q \in (0, 1/(c \log n))$. For every $i \in [0, k - 2]$ the size of $B_i(u)$ is $O(q^{-1} \log n)$ with high probability. The size of $B_{k-1}(u)$ is $O(n(q \log n)^{k-1})$, whp.*

The different bound on $|B_{k-1}(u)|$ is not really relevant for the ADO of Thorup and Zwick, as they set $q = n^{-1/k}$ and both bounds coincide at the same value. In our case, however, it raises a non trivial obstacle on the way to obtain efficient algorithms for sparse graphs. We will elaborate more on this issue later on.

The ADO is composed of an hierarchy \mathcal{S}_k^p , where $p = n^{-1/k} c \log n$, and $S_k = \emptyset$. For every $u \in V$ and $i \in [0, k]$, $p_i(u)$ and $B_i(u)$ are computed and saved in the data structure. Thorup and Zwick defined also *clusters* $C_i(w) = \{u \mid w \in B_i(u)\}$, for every $w \in S_i \setminus S_{i+1}$. The query algorithm is presented in Algorithm 1.

³ Notice that in the degenerate case of a path of two edges either $u = a$ or $b = v$

Algorithm 1 $\text{DIST}(u, v)$.

```

 $i \leftarrow 0$ ;
while  $p_i(u) \notin B_i(v)$  do
     $i \leftarrow i + 1$ ;
    swap  $u$  and  $v$ ;
end
return  $d(u, p_i(u)) + d(v, p_i(u))$ ;

```

Algorithm 2 $\text{DIST}(u, v, r)$.

```

 $i \leftarrow r$ ;
while  $p_i(u) \notin B_i(v)$  do
     $i \leftarrow i + 1$ ;
    swap  $u$  and  $v$ ;
end
return  $d(u, p_i(u)) + d(v, p_i(u))$ ;

```

For the purpose of our new algorithms we generalize the query of Thorup and Zwick's ADO (see Algorithm 2) by adding an additional input value r . The search is for the first $i \geq r$, for which $p_i(u) \in B_i(v)$, alternating between u and v after each iteration. Let $f(u, v, r)$ be the value of i when $\text{DIST}(u, v, r)$ ends.

► **Lemma 7.** *Let $u, v \in V$, $r \in [0, k-1]$ and $f = f(u, v, r)$. If $f-r$ is even then $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, v)$. If $f-r$ is odd then $d_G(v, p_f(v)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, v)$.*

Proof. Consider every $i \in [r, f-1]$. If $i-r$ is even then $p_i(u) \notin B_i(v)$ and from bunch definition together with the triangle inequality we get $d_G(v, p_{i+1}(v)) \leq d_G(v, p_i(u)) \leq d_G(u, v) + d_G(u, p_i(u))$. Similarly, if $i-r$ is odd then $p_i(v) \notin B_i(u)$ and $d_G(u, p_{i+1}(u)) \leq d_G(u, p_i(v)) \leq d_G(u, v) + d_G(v, p_i(v))$. This implies that for even $i-r$ we have $d_G(v, p_{i+1}(v)) \leq (i+1-r) \cdot d_G(u, v) + d_G(u, p_r(u))$ and for odd $i-r$ we have $d_G(u, p_{i+1}(u)) \leq (i+1-r) \cdot d_G(u, v) + d_G(u, p_r(u))$.

Thus, if $f-r$ is even we have $d_G(u, p_f(u)) \leq (f-r) \cdot d_G(u, v) + d_G(u, p_r(u))$ and if $f-r$ is odd we have $d_G(v, p_f(v)) \leq (f-r) \cdot d_G(u, v) + d_G(u, p_r(u))$.

As before we consider two subcases. The first is that r is even and the second is that r is odd.

In case that r is even then we let $f = f(u, v, r)$. ◀

3 A generalized scheme for APASP algorithms

3.1 Improving the pivots data

Baswana and Kavitha [4] presented a simple algorithm that given an hierarchy \mathcal{S}_k^p computes the distance between every pair of vertices $(s, v) \in S_i \times V$ in the graph $(V, E_{S_{i+1}} \cup E_s)$, where $i \in [0, k-1]$, in expected running time of $\tilde{O}(n^{2+1/k})$.

In our new PIVOT-DIST algorithm we compute the bunches of every $u \in V$ with respect to hierarchy \mathcal{S}_k^p . We also change the way distances are computed for the pairs $(s, v) \in S_i \times V$ in the graph $(V, E_{S_{i+1}} \cup E_s)$ by using more edges and in particular edges which are not necessarily in E .

We define a new graph $G_{i+1}(s) = (V, H_{i+1}(s))$, for every $i \in [0, k-1]$ and $s \in S_i$ and run Dijkstra's algorithm for s in $G_{i+1}(s)$. The graph $G_{i+1}(s)$ is constructed as follows. The edge set $H_{i+1}(s)$ is initialized with $E_{S_{i+1}}$. The first change is that we add to $H_{i+1}(s)$ the set of edges $E(1/(p^{i+1}))$.

The second change is that for every $s \in S_i$ the set $H(s)$ contains an edge between s and every $u \in V$. The weight of every $(s, v) \in H(s) \setminus E_s$ is initially set to ∞ . Every $(s, v) \in H(s) \cap E_s$ already has a weight. Next, we ensure in a loop that the weight of $(s, v) \in H(s)$ is at most $\min\{\mathbf{w}(s, v), \min_{u \in C_i(s) \cap N(v)} \{d_G(s, u) + \mathbf{w}(u, v)\}, \min_{u \in N(v) \wedge p_i(u)=s} \{d_G(s, u) + \mathbf{w}(u, v)\}\}$ and add $H(s)$ to $H_{i+1}(s)$.

As mentioned above we perform these computations using algorithm PIVOT-DIST. The additional data computed by PIVOT-DIST for the pairs $(s, v) \in S_i \times V$ enables us later on to perform more queries when computing the approximate distance between pairs $u, v \in V$.

■ **Algorithm 3** PIVOT-DIST(\mathcal{S}_k^p).

```

 $M[i, j] \leftarrow \infty$ , for every  $(i, j) \in [n] \times [n]$ ;
for  $i \leftarrow 0$  to  $k - 1$  do
    // Phase 1: Adding shortcut edges
    foreach  $s \in S_i \setminus S_{i+1}$  do
         $H(s) \leftarrow \{(s, v) \mid v \in V\}$ ;
        foreach  $e \in H(s) \setminus E_s$  do  $w(e) \leftarrow \infty$ ;
    end
    // Phase 2: Computing  $d_G(u, S_i)$ ,  $p_i(u)$ ,  $B_i(u)$  and updating edge weights.
    foreach  $u \in V$  do
        compute  $d_G(u, S_i)$ ,  $p_i(u)$  and  $B_i(u)$ ;
         $M[p_i(u), u] \leftarrow d_G(u, S_i)$ ,  $M[u, p_i(u)] \leftarrow d_G(u, S_i)$ ;
         $w(p_i(u), u) = d_G(u, S_i)$ ;
        (1) foreach  $(u, v) \in E_u$  do
             $w(p_i(u), v) \leftarrow \min\{w(p_i(u), v), d_G(p_i(u), u) + w(u, v)\}$ ;
            foreach  $s \in B_i(u)$  do  $M[s, u] \leftarrow d_G(u, s)$ ,  $M[u, s] \leftarrow d_G(u, s)$ ;
            (2) foreach  $(u, v) \in E_u$  do
                foreach  $s \in B_i(u)$  do  $w(s, v) \leftarrow \min\{w(s, v), d_G(s, u) + w(u, v)\}$ ;
            end
        end
    end
    // Phase 3: Computing shortest paths for the vertices of  $S_i$ 
    foreach  $s \in S_i$  do
         $G_{i+1}(s) = (V, E_{S_{i+1}} \cup E(1/(p^{i+1})) \cup H(s))$ ;
        run Dijkstra's algorithm from  $s$  in  $G_{i+1}(s)$ ;
        foreach  $u \in V$  do
             $M[s, u] \leftarrow \min\{M[s, u], d_{G_{i+1}(s)}(s, u)\}$ ,
             $M[u, s] \leftarrow \min\{M[u, s], d_{G_{i+1}(s)}(s, u)\}$ 
        end
    end
end
return  $M$ ;

```

A pseudocode of PIVOT-DIST(\mathcal{S}_k^p) for computing distance information for $(s, v) \in S_i \times V$, for every $i \in [0, k - 1]$, $s \in S_i$ and $v \in V$, is presented in Algorithm 3. A matrix M of size $n \times n$ is initialized with ∞ in every entry. We iterate on i from 0 to $k - 1$. For every i we have three phases. In the first phase we iterate on the set S_i . For every $s \in S_i \setminus S_{i+1}$ the set $H(s)$ is initialized with edges between s and every $v \in V$. For every $(s, v) \in H(s) \setminus E_s$ we set $w(s, v)$ to ∞ . In the second phase for every $u \in V$ we compute $p_i(u)$, $d_G(u, S_i)$ and $B_i(u)$. We then set $M[p_i(u), u] = d_G(u, S_i)$ and $w(p_i(u), u) = d_G(u, S_i)$. Notice that here we update the weight of edges in $H(s)$, some might be in the original graph and some might not. In line (1) we scan the edges of u and for each $(u, v) \in E_u$ we set $w(p_i(u), v) = \min\{w(p_i(u), v), d_G(p_i(u), u) + w(u, v)\}$. We set $M[s, u] = d_G(u, s)$, for every

$s \in B_i(u)$. Notice that by this we ensure that we have for every $u \in V$ the distance between u and every vertex in its bunches, according to the bunch definition of Thorup and Zwick's ADO. In line (2) we scan the edges of u and the vertices of $B_i(u)$ and for every $(u, v) \in E_u$ and $s \in B_i(u)$ we set $\mathbf{w}(s, v) = \min\{\mathbf{w}(s, v), d_G(s, u) + \mathbf{w}(u, v)\}$.

In the third phase we iterate on the vertices of S_i . We compute the distance from every $s \in S_i$ to every vertex in V in the graph $G_{i+1}(s)$ by running Dijkstra's algorithm from s and update M accordingly. The matrix M is returned as the output. Next, we analyze the running time of Algorithm 3.

► **Lemma 8.** *PIVOT-DIST(S_k^p) runs in $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$ expected time.*

Proof. Initializing M takes $O(n^2)$ time. In the first phase of the loop on i we scan $S_i \setminus S_{i+1}$ and for every $s \in S_i \setminus S_{i+1}$ we add an edge between s and every vertex of V to the set $H(s)$. We then update the weight. This costs $O(|S_i \setminus S_{i+1}|n)$. The total cost of the first phase for every $i \in [0, k-1]$ is $O(n^2)$.

In the second phase we compute $p_i(u)$, $d_G(u, S_i)$ and $B_i(u)$ for every $u \in V$. It is easy to compute $p_i(u)$ and $d_G(u, S_i)$, for every $u \in V$, in $\tilde{O}(m)$ time by connecting a dummy vertex only to the vertices of S_i and running Dijkstra's algorithm from the dummy vertex.

Thorup and Zwick [26] showed that computing $B_i(u)$ for every $u \in V$ takes $\tilde{O}(\sum_{u \in V} |B_i(u)| \cdot |N(u)|)$ time (Section 4.3 in [26]). For $i \in [0, k-2]$ this results in a running time of $\tilde{O}(m \cdot p^{-1})$, since from Lemma 6 we have $B_i(u) = \tilde{O}(1/p)$. The cost of computing $B_{k-1}(u)$, for every $u \in V$, is $\tilde{O}(m \cdot np^{k-1})$, since from Lemma 6 we have $B_{k-1}(u) = \tilde{O}(np^{k-1})$.

For every i , the cost of line (1) is $\tilde{O}(\sum_{u \in V} |N(u)|) = \tilde{O}(m)$ and the cost of line (2) is $\tilde{O}(\sum_{u \in V} |B_i(u)| \cdot |N(u)|)$. Thus, the total cost of the second phase is $\tilde{O}(km \cdot p^{-1} + m \cdot np^{k-1})$.

In the third phase we run Dijkstra's algorithm from every $s \in S_i$ in $G_{i+1}(s)$. The set of edges of $G_{i+1}(s)$ is $E_{S_{i+1}} \cup E(1/(p^{i+1})) \cup H(s)$. The set $H(s)$ is of size $O(n)$. The set $E(1/(p^{i+1}))$ is of size $O(n/(p^{i+1}))$.

Consider now the set $E_{S_{i+1}}$, for $i < k-1$. The probability of a vertex to be in S_{i+1} is $\tilde{O}(p^{i+1})$. Applying Lemma 3 we get that the expected size of the set $E_{S_{i+1}}$ is $O(n/(p^{i+1}))$.

We get that the cost of the third phase for every integer $i \in [0, k-2]$ is $\tilde{O}(|S_i| \cdot n/p^{i+1})$. Since the expected size of S_i is $\tilde{O}(np^i)$ we get a bound of $\tilde{O}(\sum_{i=0}^{k-2} (n \cdot p^i \cdot n/p^{i+1})) = \tilde{O}(n^2 + (k-1)n^2p^{-1})$.

When $i = k-1$ we cannot apply Lemma 3 to bound the size of E_{S_k} since $S_k = \emptyset$, thus, we bound the cost of running Dijkstra's algorithm from every $s \in S_{k-1}$ in $G_k(s)$ with $\tilde{O}(|S_{k-1}|m) = \tilde{O}(n \cdot p^{k-1}m)$. We get a running time of $O(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$. ◀

In the next Lemma we summarize several properties of the matrix M returned by PIVOT-DIST. These properties are ensured by phase 2.

► **Lemma 9.** *Let $i \in [0, k-1]$, let $s \in S_i$ and let $u \in V$.*

1. *If $s \in \{p_i(u)\} \cup B_i(u)$ then $M[s, u] = d_G(s, u)$*
2. *If $s \in \{p_i(u)\} \cup B_i(u)$ and $(u, v) \in E$ then $M[s, v] \leq d_G(s, u) + \mathbf{w}(u, v)$*

Proof. The proof easily follows from the execution of phase 2. ◀

We finish this section with a Lemma that summarizes several additional properties that follow from the structure of the graph $G_{i+1}(s)$, where $s \in S_i$ and $i \in [0, k-1]$. These properties are ensured by phase 3 of PIVOT-DIST.

► **Lemma 10.** *Let $u, v \in V$ and let $P(u, v)$ be a shortest path between u and v with at least two edges. Let $s \in S_i$ and let $p_i(u) = s$, where $i \in [0, k-1]$.*

1. *If $P(u, v)$ is in $G_{i+1}(s)$ then $M[s, v] \leq d_G(s, u) + d_G(u, v)$.*
2. *If $P(u, v)$ is not in $G_{i+1}(s)$ then $i \leq k-2$.*
3. *If $P(u, v)$ is not in $G_{i+1}(s)$ and $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$ overlap, where $(a, b) \in P(u, v)$ is the edge that connects them, then $(a, b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$.*

Proof.

1. The claim follows since $P(u, v)$ is in $G_{i+1}(p_i(u))$, $(p_i(u), u) \in H(p_i(u))$ and $\mathbf{w}(p_i(u), u) = d_G(p_i(u), u)$.
2. Assume towards a contradiction that $i = k-1$. The graph $G_k(p_{k-1}(u))$ contains the edges E_{S_k} and since $S_k = \emptyset$ we have $E_{S_k} = E$. Thus, $G_k(p_{k-1}(u))$ contains E and in particular, $P(u, v)$ is $G_k(p_{k-1}(u))$, a contradiction.
3. Since $d_G(u, a) < d_G(u, p_{i+1}(u))$ and $d_G(v, b) < d_G(v, p_{i+1}(v))$ it follows from Lemma 2 that $P(u, a)$ and $P(v, b)$ are in $E_{S_{i+1}}$ and thus in $G_{i+1}(p_i(u))$. Since $P(u, v)$ is not in $G_{i+1}(p_i(u))$ it must be that $(a, b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$. ◀

3.2 A general scheme

We turn now to describe our new and general scheme for APASP algorithm. We denote this algorithm with APASP. Our new algorithm is obtained by using $\text{PIVOT-DIST}(\mathcal{S}_k^p)$ to obtain better estimations in several important cases. These better estimations are then combined with a procedure similar to the query of Thorup and Zwick distance oracles. The main challenge is in the analysis of the approximation factor.

The algorithm gets as an input a weighted undirected graph G and an hierarchy \mathcal{S}_k^p . The algorithm returns a matrix M of approximate distances.

The algorithm works as follows. We start by initializing the matrix M . For every $(i, j) \in E$ we set $M[i, j]$ to $\mathbf{w}(i, j)$ and for every $(i, j) \notin E$ we set $M[i, j]$ to ∞ . Next, we run $\text{PIVOT-DIST}(\mathcal{S}_k^p)$ and update matrix M with the result. Finally, we scan for every $u, v \in V$ the vertices $p_i(u)$, for every $i \in [0, k-1]$ and the vertices in $B_i(u)$, for every $i \in [0, k-2]$. We update $M[u, v]$ if we find a vertex w such that $M[u, w] + M[v, w] < M[u, v]$. Notice that we are not scanning the vertices of $B_{k-1}(u)$ deliberately. This caveat is because the size of $B_{k-1}(u)$ is $\tilde{O}(np^{k-1})$, for every $u \in V$, thus scanning these bunches is prohibited if we like to obtain an algorithm with an efficient running time in sparse graphs. A careful case analysis of odd and even values of k allows us to avoid scanning $B_{k-1}(u)$ without affecting the approximation factor. A pseudocode of APASP is presented in Algorithm 4.

We now analyze the approximation of the algorithm.

► **Lemma 11.** *The output M of $\text{APASP}(G, \mathcal{S}_k^p)$ satisfies $d_G(u, v) \leq M[u, v] \leq 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$.*

Proof. Let $u, v \in V$. Let $P(u, v)$ be a shortest path between u and v . If $P(u, v)$ has a single edge then $M[u, v] = \mathbf{w}(u, v)$ since we set $M[u, v]$ to $\mathbf{w}(u, v)$ in APASP. Thus, we can assume that $P(u, v)$ has at least 2 edges. Let $i \in [0, k-1]$ be the largest index such that $d_G(u, S_i) + d_G(v, S_i) \leq d_G(u, v)$. Such an index must exist since $S_0 = V$ which implies that $d_G(u, S_0) + d_G(v, S_0) = 0 \leq d_G(u, v)$.

⁴ If $P(u, v)$ has two edges then the claim hold since either $v = b$ or $u = a$.

Algorithm 4 APASP(G, \mathcal{S}_k^p).

```

foreach  $(i, j) \in [n] \times [n]$  do  $M[i, j] \leftarrow \infty$ ;
foreach  $(i, j) \in E$  do  $M[i, j] \leftarrow \mathbf{w}(i, j)$ ;
 $M \leftarrow \min\{M, \text{PIVOT-DIST}(\mathcal{S}_k^p)\}$ ;
foreach  $u \in V$  do
  for  $i \leftarrow 0$  to  $k - 1$  do
    foreach  $v \in V$  do
      (1)  $M[u, v] \leftarrow \min\{M[u, v], M[p_i(u), u] + M[p_i(u), v], M[v, u]\}$ ;
      if  $i \leq k - 2$  then
        (2) foreach  $w \in B_i(u)$  do
           $M[u, v] \leftarrow \min\{M[u, v], M[w, u] + M[w, v], M[v, u]\}$ 
        end
      end
    end
  end
end
return  $M$ ;

```

Assume, wlog, that $d_G(u, S_i) \leq d_G(v, S_i)$. If $P(u, v)$ is in $G_{i+1}(p_i(u))$ it follows from Lemma 9(i) and from Lemma 10(i) that after updating M with the result of PIVOT-DIST $M[p_i(u), u] = d_G(p_i(u), u)$ and $M[p_i(u), v] \leq d_G(p_i(u), u) + d_G(u, v)$. In line (1) of APASP we update $M[u, v]$ if needed, therefore, it is guaranteed that $M[u, v] \leq 2d_G(p_i(u), u) + d_G(u, v) \leq 2d_G(u, v)$.

Consider now the case that $P(u, v)$ is not in $G_{i+1}(p_i(u))$. From Lemma 10(ii) it follows that $i < k - 1$. Since i is the largest index for which we have $d_G(u, S_i) + d_G(v, S_i) \leq d_G(u, v)$ it follows that $d_G(u, S_{i+1}) + d_G(v, S_{i+1}) > d_G(u, v)$, which implies that $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$ overlap. Let $(a, b) \in P(u, v)$ be the edge that connects $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$. From Lemma 10(iii) it follows that $(a, b) \notin E_a(1/p^{i+1})$ and $(a, b) \notin E_b(1/p^{i+1})$. Therefore, for the rest of the proof we can assume that $P(u, v)$ has at least 2 edges, $i < k - 1$, $P(u, a)$ is in $G(V, E_{S_{i+1}})$, $P(v, b)$ is in $G(V, E_{S_{i+1}})$ and $(a, b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$.

Next, we will prove two different bounds on $M[u, v]$.

▷ **Claim 12.** $M[u, v] \leq \min\{3d_G(u, v) - 2d_G(b, v), 3d_G(u, v) - 2d_G(a, u)\}$.

Proof. Let $j > i + 1$ be the smallest index for which $(a, b) \in E(1/p^j)$. If $j \geq k$ we set j to k . Assume that $(a, b) \in E_a(1/p^j)$ and $(a, b) \in E_b(1/p^{j'})$, where $j' \geq j$. If $j = k$ we set j' to k . By definition, the sets $N(a, 1/p^{j-1})$ and $N(b, 1/p^{j-1})$ are of size $1/p^{j-1}$. Since vertices of V are in S_{j-1} with probability at least $p^{j-1}c \log n$, it follows that, whp, S_{j-1} contains at least one vertex from $N(a, 1/p^{j-1})$ and $N(b, 1/p^{j-1})$.

Let $x \in N(a, 1/p^{j-1}) \cap S_{j-1}$ and let $y \in N(b, 1/p^{j-1}) \cap S_{j-1}$. Since $(a, x) \in E_a(1/p^{j-1})$ and $(a, b) \notin E_a(1/p^{j-1})$ we have $\mathbf{w}(a, x) \leq \mathbf{w}(a, b)$. Similarly, since $(b, y) \in E_b(1/p^{j-1})$ and $(a, b) \notin E_b(1/p^{j-1})$ we have $\mathbf{w}(b, y) \leq \mathbf{w}(a, b)$. Thus, we get that $d_G(u, p_{j-1}(u)) \leq d_G(u, a) + \mathbf{w}(a, b) \leq d_G(u, b)$ and $d_G(v, p_{j-1}(v)) \leq d_G(v, b) + \mathbf{w}(a, b) \leq d_G(v, a)$.

Now since $(a, b) \in E_a(1/p^j)$ it follows that the path $P(u, v)$ is in $G_j(p_{j-1}(u))$ and $G_j(p_{j-1}(v))$, when $j < k$. When $j = k$ we have $E_{S_k} = E$ and $P(u, v)$ is in $G_k(p_{k-1}(u))$. It follows from Lemma 10(i) that $M[p_{j-1}(u), v] \leq d_G(p_{j-1}(u), u) + d_G(u, v)$ and $M[p_{j-1}(v), u] \leq d_G(p_{j-1}(v), v) + d_G(u, v)$.

Thus, after line (1) of APASP is executed for the pair u, v and for the pair v, u we have $M[u, v] = M[v, u] \leq \min\{2d_G(u, p_{j-1}(u)) + d_G(u, v), 2d_G(v, p_{j-1}(v)) + d_G(u, v)\}$. We get:

$$\begin{aligned} M[v, u] = M[u, v] &\leq \min\{2d_G(u, p_{j-1}(u)) + d_G(u, v), 2d_G(v, p_{j-1}(v)) + d_G(u, v)\} \\ &\leq \min\{d_G(u, v) + 2d_G(u, b), d_G(u, v) + 2d_G(v, a)\} \\ &= \min\{3d_G(u, v) - 2d_G(b, v), 3d_G(u, v) - 2d_G(a, u)\}, \end{aligned}$$

since $d_G(u, p_{j-1}(u)) \leq d_G(u, b)$ and $d_G(v, p_{j-1}(v)) \leq d_G(v, a)$ and since $d_G(u, b) = d_G(u, v) - d_G(b, v)$ and $d_G(v, a) = d_G(u, v) - d_G(a, u)$. \triangleleft

We now turn to prove a second bound on $M[u, v]$.

\triangleright **Claim 13.** $M[u, v]$ is bounded either by (i) $2 \cdot (k-1)d_G(u, a) + d_G(u, v)$ or by (ii) $2 \cdot (k-1)d_G(v, b) + d_G(u, v)$

Proof. We consider the pair of vertices u and a and the pair of vertices v and b .

Let r be the largest index such that $a \notin B_{S_r}(u)$ and $u \notin B_{S_r}(a)$. Let r' be the largest index such that $b \notin B_{S_{r'}}(v)$ and $v \notin B_{S_{r'}}(b)$. Since $S_0 = V$ it follows that $B_{S_0}(x) = \{x\}$, for every $x \in V$. Thus, we have $a \notin B_{S_0}(u)$ and $u \notin B_{S_0}(a)$ and also $b \notin B_{S_0}(v)$ and $v \notin B_{S_0}(b)$.

We assume that $r \geq r'$ and show that $M[u, v] \leq 2 \cdot (k-1)d_G(u, a) + d_G(u, v)$. If $r' \geq r$ then a symmetric proof shows that $M[u, v] \leq 2 \cdot (k-1)d_G(v, b) + d_G(u, v)$. In the degenerate case that $P(u, v)$ has only two edges and $b = v$ we set $r' = r$. The proof below works for this case as well and hence $M[u, v] \leq 2 \cdot (k-1)d_G(u, a) + d_G(u, v)$. If $a = u$ a symmetric proof shows that $M[u, v] \leq 2 \cdot (k-1)d_G(v, b) + d_G(u, v)$.

Recall that we are in the case that $d_G(u, a) < d_G(u, p_{i+1}(u))$ and $d_G(v, b) < d_G(v, p_{i+1}(v))$, thus we have $a \in B_{S_{i+1}}(u)$ and $b \in B_{S_{i+1}}(v)$. This implies that $r \leq i < k-1$ and $r' \leq i < k-1$, since $i < k-1$.

From the definition of r it follows that either $a \in B_{S_{r+1}}(u)$ or $u \in B_{S_{r+1}}(a)$. If $a \in B_{S_{r+1}}(u)$ then $d_G(u, a) < d_G(u, p_{r+1}(u))$ and it follows from Lemma 2 that $P(u, a) \in E_{S_{r+1}}$. Similarly, if $u \in B_{S_{r+1}}(a)$ then $d_G(u, a) < d_G(a, p_{r+1}(a))$ and it follows from Lemma 2 that $P(u, a) \in E_{S_{r+1}}$. From symmetrical arguments we get that $P(v, b) \in E_{S_{r'+1}}$. Since $E_{S_j} \subseteq E_{S_{j+1}}$, for every $j \in [0, k-1]$, we have and $P(u, a) \cup P(v, b) \subseteq E_{S_{q+1}}$, for every $q \geq r$.

Next, we distinguish between odd and even values of k . Assume first that k is odd. This implies that $k-1$ is even. We now consider two subcases. The first is that r is even and the second is that r is odd. In case that r is even then we let $f = f(u, a, r)$.

From Lemma 7 it follows that if $f-r$ is even then we have $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, a)$ and if $f-r$ is odd then $d_G(a, p_f(a)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, a)$.

We proceed by considering these two scenarios.

1. **Even $f-r$.** In this case $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, a)$ and $p_f(u) \in B_f(a)$. It follows from Lemma 9(ii) that after Phase 2 of PIVOT-DIST $M[p_f(u), b] \leq d_G(p_f(u), a) + \mathbf{w}(a, b)$. When executing in Phase 3 of PIVOT-DIST Dijkstra's algorithm from $p_f(u)$ in $G_{f+1}(p_f(u))$ with edge set $E_{S_{f+1}} \cup E(1/(p^{f+1})) \cup H(p_f(u))$ the weight of the edge $(p_f(u), b) \in H(p_f(u))$ is $M[p_f(u), b] \leq d_G(p_f(u), a) + \mathbf{w}(a, b)$ and $P(v, b) \subseteq E_{S_{f+1}}$. Thus, we get that $M[p_f(u), v] \leq d_G(p_f(u), a) + d_G(a, v) \leq d_G(p_f(u), u) + d_G(u, v)$. In line (1) of APASP we update $M[u, v]$ so that it is at most $M[p_f(u), u] + M[p_f(u), v]$. From Lemma 9(i) it follows that $M[p_f(u), u] = d_G(p_f(u), u)$ and since $M[p_f(u), v] \leq d_G(p_f(u), u) + d_G(u, v)$ we get that $M[u, v] \leq 2 \cdot d_G(p_f(u), u) + d_G(u, v)$. (See
Combining this with the fact that $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, a)$ we get that for even $f-r$ we have $M[u, v] \leq 2 \cdot (d_G(u, p_r(u)) + (f-r)d_G(u, a)) + d_G(u, v)$. Recall also that $f \leq k-1$. In the degenerate case of $r = 0$ we have $p_0(u) = u$ and $M[u, v] \leq 2 \cdot (k-1)d_G(u, a) + d_G(u, v)$.

For the case that $r \geq 1$ since $a \notin B_{S_r}(u)$ we have $d_G(u, p_r(u)) \leq d_G(u, a)$. Using this we get $M[u, v] \leq 2 \cdot ((f - r + 1)d_G(u, a)) + d_G(u, v) \leq 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$.

2. **Odd $f - r$.** In this case $d_G(a, p_f(a)) \leq d_G(u, p_r(u)) + (f - r)d_G(u, a)$ and $p_f(a) \in B_f(u)$. It follows from Lemma 9(ii) that after Phase 2 of PIVOT-DIST $M[p_f(a), b] \leq d_G(p_f(a), a) + \mathbf{w}(a, b)$. When executing in Phase 3 of PIVOT-DIST Dijkstra's algorithm from $p_f(a)$ in $G_{f+1}(p_f(a))$ with edge set $E_{S_{f+1}} \cup E(1/(p^{f+1})) \cup H(p_f(a))$ the weight of the edge $(p_f(a), b) \in H(p_f(a))$ is $M[p_f(a), b] \leq d_G(p_f(a), a) + \mathbf{w}(a, b)$ and $P(v, b) \subseteq E_{S_{f+1}}$. Thus, we get that $M[p_f(a), v] \leq d_G(p_f(a), a) + d_G(a, v)$.

Consider the execution of line (2) in APASP for vertices v, u and a and the index f . Notice that this line is only executed for indices $i \leq k - 2$. In our case we have $p_f(a) \in B_f(u)$ and since r is even and $f - r$ is odd, f must be odd. Since $k - 1$ is even we have $f \leq k - 2$. Thus line (2) is executed and we update $M[u, v]$ so that it is at most $M[p_f(a), u] + M[p_f(a), v]$. Since $p_f(a) \in B_f(u)$ it follows from Lemma 9(i) that $M[p_f(a), u] = d_G(p_f(a), u) \leq d_G(u, a) + d_G(p_f(a), a)$ and since $M[p_f(a), v] \leq d_G(p_f(a), a) + d_G(a, v)$ we get that $M[u, v] \leq 2 \cdot d_G(p_f(a), a) + d_G(u, v)$.

Combining this with the fact that $d_G(a, p_f(a)) \leq d_G(u, p_r(u)) + (f - r)d_G(u, a)$ we get that for odd $f - r$ we have $M[u, v] \leq 2 \cdot (d_G(u, p_r(u)) + (f - r)d_G(u, a)) + d_G(u, v)$. Recall also that $f \leq k - 1$.

In the degenerate case of $r = 0$ we have $p_0(u) = u$ and $M[u, v] \leq 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$.

For the case that $r \geq 1$ since $a \notin B_{S_r}(u)$ we have $d_G(u, p_r(u)) \leq d_G(u, a)$. Using this we get $M[u, v] \leq 2 \cdot ((f - r + 1)d_G(u, a)) + d_G(u, v) \leq 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$.

Consider now the second case in which r is odd and let $f = f(a, u, r)$.

1. **Odd $f - r$.** In this case $d_G(u, p_f(u)) \leq d_G(a, p_r(a)) + (f - r)d_G(u, a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ and even $f - r$.
2. **Even $f - r$.** In this case $d_G(a, p_f(a)) \leq d_G(a, p_r(a)) + (f - r)d_G(u, a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ and odd $f - r$.

Assume now that k is even. This implies that $k - 1$ is odd.

1. **Odd $f - r$.** In this case $d_G(u, p_f(u)) \leq d_G(a, p_r(a)) + (f - r)d_G(u, a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ and even $f - r$.
2. **Even $f - r$.** In this case $d_G(a, p_f(a)) \leq d_G(a, p_r(a)) + (f - r)d_G(u, a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ and odd $f - r$.

Consider now the second case in which r is odd and let $f = f(u, a, r)$.

1. **Even $f - r$.** In this case $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f - r)d_G(u, a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ even $f - r$.
2. **Odd $f - r$.** In this case $d_G(a, p_f(a)) \leq d_G(u, p_r(u)) + (f - r)d_G(u, a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r , $f = f(u, a, r)$ and odd $f - r$. \triangleleft

We now combine the two bounds to complete the proof. From Claim 13 it follows that either $M[u, v] \leq 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$ or $M[u, v] \leq 2 \cdot (k - 1)d_G(v, b) + d_G(u, v)$. Assume that $M[u, v] \leq 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$. From Claim 12 it follows that $M[u, v] \leq \min\{3d_G(u, v) - 2d_G(b, v), 3d_G(u, v) - 2d_G(a, u)\}$. Thus, $M[u, v] \leq \min\{2 \cdot (k - 1)d_G(u, a) + d_G(u, v), 3d_G(u, v) - 2d_G(a, u)\}$. Let $X = 2 \cdot (k - 1)d_G(u, a) + d_G(u, v)$ and let $Y = 3d_G(u, v) - 2d_G(a, u)$. When $d_G(u, a) = d_G(u, v)/k$, we have $X = Y = 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$. When $d_G(u, a) < d_G(u, v)/k$ we have $X < 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$. When $d_G(u, a) > d_G(u, v)/k$ we have $Y < 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$. Since $M[u, v] \leq \min\{X, Y\}$ we get that $M[u, v] \leq 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$.

Consider the degenerate case in which $P(u, v)$ has only two edges, and assume that $b = v$. From Claim 13 it follows that $M[u, v] \leq 2 \cdot (k-1)d_G(u, a) + d_G(u, v)$. From Claim 12 it follows that $M[u, v] \leq 3d_G(u, v) - 2d_G(a, u)$. Thus, from the same arguments as above, $M[u, v] \leq 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$. \blacktriangleleft

We now turn to analyze the running time of the algorithm.

► **Lemma 14.** *The expected running time of $\text{APASP}(G, \mathcal{S}_k^p)$ is $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$*

Proof. Initializing M takes $O(n^2)$ time. From Lemma 8 it follows that the call to PIVOT-DIST takes $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$ time. Finally, for every $u \in V$, $i \in [0, k-2]$ and $v \in V$ we do $|B_i(u)| + 1$ read operations to matrix M . This takes $\tilde{O}(n^2 + k(n^2p^{-1}))$ time, since $|B_i(u)| = \tilde{O}(p^{-1})$, for $i \in [0, k-2]$. \blacktriangleleft

We can now prove:

The approach presented so far powerful enough on its own to obtain the following non trivial generalization, which already improves the general scheme presented by Cohen and Zwick [9].

► **Theorem 15.** *For every integer $k \geq 2$, there is an APASP algorithm with expected running time of $\tilde{O}(kn^{2-1/k}m^{1/k})$ and multiplicative approximation of $2 + \frac{k-2}{k}$.*

Proof. We run APASP with the hierarchy \mathcal{S}_k^p . From Lemma 14 it follows that the running time is $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$. Setting $n^2p^{-1} = mnp^{k-1}$ we get $p = \left(\frac{n^2}{mn}\right)^{1/k}$. Thus, the running time is $\tilde{O}(kn^{2-1/k}m^{1/k})$. From Lemma 11 it follows that the multiplicative approximation is $2 + \frac{k-2}{k}$. \blacktriangleleft

4 A faster scheme

In this section we first present augmented hierarchies. This is basically the first part of the 2-approximation algorithm of Baswana and Kavitha [4]. We provide a tighter analysis that relaxes the requirements for getting a 2-approximation. We then present the idea of mixed hierarchies, a combination of a regular hierarchy with the last set of the augmented hierarchy. This allows us to connect our algorithm from Section 3, that worked with a base set V , to the first part of the 2-approximation algorithm, by forcing the second set of a mixed hierarchy to be the last set of the augmented hierarchy. We end by proving Theorem 1.

4.1 Augmented hierarchy with $\hat{S}_k \neq \emptyset$

Thorup and Zwick [25] showed that if a set $S \subseteq V$ is constructed by a careful recursive sampling procedure then the maximum size of every cluster is bounded as well. They proved:

► **Lemma 16** ([25]). *Given a parameter p , we can compute a set S of size $\tilde{O}(np)$ in $\tilde{O}(mp^{-1})$ expected time such that, $|C_S(w)| = O(1/p)$ for every vertex $w \in V \setminus S$, and $|B_S(v)| = O(1/p)$ for every $v \in V$.*

Following Baswana and Kavitha [4] we will extend the vertex hierarchy to an *augmented hierarchy*, denoted with $\hat{\mathcal{S}}_k^p$. Let \mathcal{S}_k^p be a regular hierarchy with $S_0 = V$ and $S_k = \emptyset$. Let S be a set computed using Lemma 16 with parameter p^k . In $\hat{\mathcal{S}}_k^p$ we have $\hat{S}_i = S_i \cup S$, for every $i \in [0, k]$. In particular, we have $\hat{S}_k = S$. We also set $\hat{S}_{k+1} = \emptyset$. We refer to S as the augmenting set of the hierarchy.

The pivots data that we compute for an augmented hierarchy \hat{S}_k^p is computed using algorithm AUG-PIVOT-DIST. Algorithm AUG-PIVOT-DIST differs from PIVOT-DIST by an additional special phase that is added between the first phase and the second phase and by avoiding the explicit computation of bunches in phase 2. The special phase is devoted for the set \hat{S}_k . For every $u \in V$, we compute $B_{\hat{S}_k}(u)$ and $C_{\hat{S}_k}(u)$. For every $a \in B_{\hat{S}_k}(u) \cup C_{\hat{S}_k}(u)$ and $i \in [0, k]$ we scan the edges of a and update the weight of the edge between $p_i(u)$ and every $b \in N(a)$ so that it will be at most $d_G(p_i(u), u) + d_G(u, a) + \mathbf{w}(a, b)$.

Next, we analyze the running time of AUG-PIVOT-DIST.

► **Lemma 17.** *If S is computed using Lemma 16 with parameter p^k then the expected running time of Algorithm AUG-PIVOT-DIST(\hat{S}_k^p) is $\tilde{O}(n^2 + (k-1)n^2p^{-1} + kmp^{-k})$*

Proof. Phase 1 remains unchanged, therefore, its cost remains $\tilde{O}(n^2)$ as in Lemma 8.

In the special phase we compute for every $u \in V$ the ball $B_S(u)$ in $\tilde{O}(\sum_{u \in V} \deg(u)|B_S(u)|)$ time as was shown by Baswana and Kavitha [4]. Since $|B_S(u)| = O(p^{-k})$ this part takes $\tilde{O}(mp^{-k})$ time. Since clusters are simply the inverse of balls we can compute them at the same cost. For every $a \in B_{\hat{S}_k}(u) \cup C_{\hat{S}_k}(u)$ and for every $i \in [0, k-1]$ we scan the edges of a and update for every $b \in N(a)$ the value of $\mathbf{w}(p_i(u), b)$, if needed. The total cost of this is $O(k \sum_{a \in V} |C_{\hat{S}_k}(a) \cup B_{\hat{S}_k}(a)| \deg(a)) = O(kmp^{-k})$. In phase 2 we do not compute bunches as before and thus the cost is only $\tilde{O}(m)$ for the computation of $p_i(u)$ and $d_G(u, S_i)$. In phase 3 we no longer have to split the analysis of the case that $i = k-1$ from the analysis of the general case. This is due to the fact that $\hat{S}_k \neq \emptyset$. Since a vertex is in \hat{S}_k with probability less than p^k we can apply Lemma 3 and get that the expected size of E_{S_k} is $O(n/(p^k))$. Therefore, phase 3 takes $\tilde{O}(n^2 + (k-1)n^2p^{-1})$ time. ◀

We use AUG-PIVOT-DIST for computing APASP in an algorithm denoted with AUG-APASP. The running time analysis of AUG-APASP stems from Lemma 17 and Lemma 14.

► **Corollary 18.** *AUG-APASP(G, \hat{S}_k^p) has $\tilde{O}(n^2 + kn^2p^{-1} + kmp^{-k})$ expected running time.*

In order to analyse the approximation produced by AUG-APASP we introduce the following definition which allows us to relax the condition required for proving a 2 approximation.

► **Definition 19.** *Let $u, v \in V$ and let $S \subseteq V$. We say that u and v are covered by balls $B_S(u)$ and $B_S(v)$ if there is a shortest path P between u and v such that $P \subseteq B_S(u) \cup B_S(v)$.*

The next Lemma follows easily from the above definition.

► **Lemma 20.** *If $B_S(u)$ and $B_S(v)$ do not cover $u, v \in V$ then $d_G(u, S) + d_G(v, S) \leq d_G(u, v)$*

Proof. As u and v are not covered by $B_S(u)$ and $B_S(v)$ every shortest path $P(u, v)$ has a vertex $w \notin B_S(u) \cup B_S(v)$. Thus, $d_G(u, S) \leq d_G(u, w)$ and $d_G(v, S) \leq d_G(v, w)$. Since $w \in P(u, v)$ we get $d_G(u, v) = d_G(u, w) + d_G(v, w)$ and $d_G(u, S) + d_G(v, S) \leq d_G(u, v)$. ◀

We are now ready to bound the approximation produced by AUG-APASP.

► **Lemma 21.** *Let \hat{S}_k^p be an augmented hierarchy with $\hat{S}_0 = V$ and $\hat{S}_k = S$. If $B_{\hat{S}_k}(u)$ and $B_{\hat{S}_k}(v)$ do not cover $u, v \in V$ then AUG-APASP(G, \hat{S}_k^p) returns a matrix M that satisfies: $d_G(u, v) \leq M[u, v] \leq 2d_G(u, v)$.*

Proof. Let $u, v \in V$. Let $i \in [0, k]$ be the smallest index such that $B_{\hat{S}_i}(u)$ and $B_{\hat{S}_i}(v)$ cover u and v . Such an index must exist since u and v are covered by $B_{\hat{S}_k}(u)$ and $B_{\hat{S}_k}(v)$. Since $B_{\hat{S}_0}(v) = \{v\}$ and $B_{\hat{S}_0}(u) = \{u\}$ and since we can assume that any shortest path between u

and v is of at least two edges we have that u and v are not covered by $B_{\hat{S}_0}(v)$ and $B_{\hat{S}_0}(u)$. Thus, $i > 0$. From Lemma 20 it follows that $d_G(u, \hat{S}_{i-1}) + d_G(v, \hat{S}_{i-1}) \leq d_G(u, v)$. Assume, wlog, that $d_G(u, \hat{S}_{i-1}) \leq d_G(v, \hat{S}_{i-1})$. Let $P(u, v)$ be a shortest path between u and v such that $P(u, v) \subseteq B_{\hat{S}_i}(u) \cup B_{\hat{S}_i}(v)$.

We can partition $P(u, v)$ into three portions. A portion $P(u, a)$ between u and a in $B_{\hat{S}_i}(u)$, a portion $P(b, v)$ between v and b in $B_{\hat{S}_i}(v)$, and an edge (a, b) . Since $d_G(u, a) < d_G(u, p_i(u))$ it follows from Lemma 2 that $P(u, a)$ is in $G(V, E_{\hat{S}_i})$ and $d_{(V, E(\hat{S}_i))}(u, a) = d_G(u, a)$. Similarly, $P(v, b)$ is in $G(V, E_{\hat{S}_i})$ and $d_{(V, E(\hat{S}_i))}(v, b) = d_G(v, b)$.

Now $B_{\hat{S}_i}(u) \subseteq B_{\hat{S}_k}(u)$. This implies that $a \in B_{\hat{S}_k}(u)$ and $\mathbf{w}(p_{i-1}(u), b)$ was updated in the special phase of AUG-PIVOT-DIST such that $\mathbf{w}(p_{i-1}(u), b) \leq d_G(p_{i-1}(u), u) + d_G(u, a) + \mathbf{w}(a, b)$. After running Dijkstra's algorithm in phase 3 of AUG-PIVOT-DIST for $p_{i-1}(u)$ in $G_i(p_{i-1}(u))$ we have $M[p_{i-1}(u), v] \leq d_G(p_{i-1}(u), u) + d_G(u, v)$. Thus, after updating the M with the result of AUG-PIVOT-DIST we have $M[p_{i-1}(u), v] \leq d_G(p_{i-1}(u), u) + d_G(u, v)$ and $M[p_{i-1}(u), u] = d_G(p_{i-1}(u), u)$. In line (1) of AUG-APASP we update $M[u, v]$ if needed, therefore, it is guaranteed that $M[u, v] \leq 2d_G(p_{i-1}(u), u) + d_G(u, v) \leq 2d_G(u, v)$. \blacktriangleleft

4.2 A mixed hierarchy

Let S be the augmenting set of an augmented hierarchy. We define a *mixed hierarchy* as follows: $\bar{S}_0 = V$, $\bar{S}_1 = S$. For $i \in [2, k-1]$ the set \bar{S}_i is constructed by picking every vertex of \bar{S}_{i-1} independently at random with probability $p \cdot c \log n$, for some constant c . The set \bar{S}_k is empty. We denote a mixed hierarchy with $\bar{S}_k^{p,q}$, where q is the parameter used to create S by Lemma 16.

We update PIVOT-DIST to handle a mixed hierarchy. We add a special phase as in AUG-PIVOT-DIST for the set \bar{S}_1 . Since $|\bar{S}_0| = n$ and $|E_{\bar{S}_1}| = O(nq^{-1})$ we do not execute phase 3 for the vertices of $\bar{S}_0 = V$ to avoid an additional cost of $\tilde{O}(n^2q^{-1})$ time. In phase 3 the set $E(1/(p^{i+1}))$ is changed to $E(1/(qp^i))$, so that it reflects the size of \bar{S}_1 . The updated algorithm is called MIX-PIVOT-DIST.

Next, we analyze the running time of MIX-PIVOT-DIST.

► **Lemma 22.** *MIX-PIVOT-DIST($\bar{S}_k^{p,q}$) has an expected running time of $\tilde{O}(n^2 + (k-2)n^2p^{-1} + mq^{-1} + mnqp^{k-2})$.*

Proof. The analysis of the first phase remains as in Lemma 8, and therefore it is $O(n^2)$. The special phase costs $\tilde{O}(mq^{-1})$ as in Lemma 17. In the second phase we compute $p_i(u)$, $d_G(u, \bar{S}_i)$ and $B_i(u)$ for every $u \in V$. It takes $\tilde{O}(m)$ time to compute $p_i(u)$ and $d_G(u, \bar{S}_i)$, for every $u \in V$. Computing $B_i(u)$ for every $u \in V$, where $i \in [2, k-2]$ costs $\tilde{O}(m \cdot p^{-1})$. Computing $B_1(u)$ for every $u \in V$ costs $\tilde{O}(m \cdot q^{-1})$ since q is the parameter used for \bar{S}_1 . Computing $B_{k-1}(u)$, for every $u \in V$, costs $\tilde{O}(m \cdot nqp^{k-2})$, since from Lemma 6 we have $B_{k-1}(u) = \tilde{O}(nqp^{k-2})$.

Line (1) costs $\tilde{O}(\sum_{u \in V} |N(u)|) = \tilde{O}(m)$ and line (2) costs $\tilde{O}(\sum_{u \in V} |B_i(u)| \cdot |N(u)|) = \tilde{O}(m \cdot (p^{-1} + q^{-1}))$.

In the third phase we run Dijkstra's algorithm from every $s \in \bar{S}_i$ in $G_{i+1}(s)$, as before. The set of edges of $G_{i+1}(s)$ is $E_{\bar{S}_{i+1}} \cup E(1/(qp^i)) \cup H(s)$. The set $H(s)$ is of size $O(n)$. The set $E(1/(qp^i))$ is of size $O(n/(qp^i))$. Consider now the set $E_{\bar{S}_{i+1}}$, for $0 < i < k-1$. The probability of a vertex to be in \bar{S}_{i+1} is $\tilde{O}(qp^i)$. Applying Lemma 3 we get that the expected size of the set $E_{\bar{S}_{i+1}}$ is $O(n/(qp^i))$. We get that the cost of the third phase for every $i \in [1, k-2]$ is $\tilde{O}(|\bar{S}_i| \cdot n/(qp^i))$. Since the expected size of \bar{S}_i is $\tilde{O}(nqp^{i-1})$ we get a bound of $\tilde{O}(\sum_{i=1}^{k-2} (n \cdot qp^{i-1} \cdot n/(qp^i)) = \tilde{O}(n^2 + (k-2)n^2p^{-1})$.

When $i = k - 1$ we cannot apply Lemma 3 to bound the size of $E_{\bar{S}_k}$ since $\bar{S}_k = \emptyset$, thus, we bound the cost of running Dijkstra's algorithm from every $s \in \bar{S}_{k-1}$ in $G_k(s)$ with $\tilde{O}(|\bar{S}_{k-1}|m) = \tilde{O}(n \cdot qp^{k-2}m)$. We get a running time of $O(n^2 + (k-2)n^2p^{-1} + mnqp^{k-2})$. ◀

MIX-APASP is algorithm APASP in which MIX-PIVOT-DIST is called instead of PIVOT-DIST. We do not execute Line (2) of APASP when $i = 0$, from the same reason we have not executed phase 3 for the vertices of $\bar{S}_0 = V$ in MIX-PIVOT-DIST. The analysis of the running time of MIX-APASP is relatively straightforward and stems from Lemma 22 and Lemma 14.

► **Corollary 23.** *MIX-APASP($G, \hat{S}_k^{p,q}$) runs in $\tilde{O}(n^2 + kn^2p^{-1} + mq^{-1} + mnqp^{k-2})$ expected time.*

We prove a variant of Lemma 11 for MIX-APASP ($G, \bar{S}_k^{p,q}$). The difference with respect to Lemma 11 is that the Lemma is proved only for vertices $u, v \in V$ that are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$. This change is required since we are not computing shortest paths for the vertices of \bar{S}_0 in phase 3 of MIX-PIVOT-DIST.

► **Lemma 24.** *MIX-APASP($G, \bar{S}_k^{p,q}$) returns a matrix M that satisfies: $d_G(u, v) \leq M[u, v] \leq 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$, for every $u, v \in V$ that are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$.*

Proof. Let $u, v \in V$. Let $P(u, v)$ be a shortest path between u and v . Let $i \in [0, k-1]$ be the largest index such that $d_G(u, \bar{S}_i) + d_G(v, \bar{S}_i) \leq d_G(u, v)$. Such an index must exist since $\bar{S}_0 = V$, which implies that $d_G(u, \bar{S}_0) + d_G(v, \bar{S}_0) = 0 \leq d_G(u, v)$. Since u and v are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$, it follows from Lemma 20 that $d_G(u, \bar{S}_1) + d_G(v, \bar{S}_1) \leq d_G(u, v)$, as well and $i \geq 1$. This implies that the analysis of the case that $P(u, v)$ is in $G_{i+1}(p_i(u))$ remains the same and is not affected by the fact that in MIX-PIVOT-DIST no data is computed in phase 3 for the set \bar{S}_0 .

The analysis of the case that the path $P(u, v)$ is not in $G_{i+1}(p_i(u))$ is not using the third phase data of \bar{S}_0 and remains the same. Thus, for the rest of the proof we can assume that $0 < i < k-1$, $P(u, a)$ is in $G(V, E_{\bar{S}_{i+1}})$, $P(v, b)$ is in $G(V, E_{\bar{S}_{i+1}})$ and $(a, b) \notin E_a(1/qp^i) \cup E_b(1/qp^i) \cup E_{\bar{S}_{i+1}}$.

The proof of Claim 12 remains the same since we only use data for pivots from \bar{S}_{j-1} , where $j > i+1$ and $i > 0$. Thus, not computing data for \bar{S}_0 in phase 3 of MIX-PIVOT-DIST has no affect.

The proof of Claim 13 remains also the same from the following reason. Recall that we focus on the pair of vertices u and a and the pair of vertices v and b . In the proof we use r which is defined to be the largest index such that $a \notin B_{\bar{S}_r}(u)$ and $u \notin B_{\bar{S}_r}(a)$ and r' which is defined to be the largest index such that $b \notin B_{\bar{S}_{r'}}(v)$ and $v \notin B_{\bar{S}_{r'}}(b)$. Recall also that since $\bar{S}_0 = V$ we have $B_{\bar{S}_0}(x) = \{x\}$, for every $x \in V$ and thus $a \notin B_{\bar{S}_0}(u)$ and $u \notin B_{\bar{S}_0}(a)$ and similarly $b \notin B_{\bar{S}_0}(v)$ and $v \notin B_{\bar{S}_0}(b)$.

From the definition of r it follows that either $a \in B_{\bar{S}_{r+1}}(u)$ or $u \in B_{\bar{S}_{r+1}}(a)$. If $a \in B_{\bar{S}_{r+1}}(u)$ then $d_G(u, a) < d_G(u, \bar{S}_{r+1})$ and it follows from Lemma 2 that $P(u, a) \in E_{\bar{S}_{r+1}}$. Similarly, if $u \in B_{\bar{S}_{r+1}}(a)$ then $d_G(u, a) < d_G(a, \bar{S}_{r+1})$ and it follows from Lemma 2 that $P(u, a) \in E_{\bar{S}_{r+1}}$.

From symmetrical arguments we get that $P(v, b) \in E_{\bar{S}_{r'+1}}$. Since $E_{\bar{S}_j} \subseteq E_{\bar{S}_{j+1}}$, for every $j \in [0, k-1]$, we have and $P(u, a) \subseteq E_{\bar{S}_{q+1}}$, and $P(v, b) \subseteq E_{\bar{S}_{q'+1}}$ for every $q \geq r$ and $q' \geq r'$, respectively.

Because we do not compute in the third phase data for the set \bar{S}_0 we deal separately with the special case that $r = r' = 0$. In such a case we have $a \in B_{\bar{S}_1}(u)$ or $u \in B_{\bar{S}_1}(a)$ and also $b \in B_{\bar{S}_1}(v)$ or $v \in B_{\bar{S}_1}(b)$ ⁵. Now since u and v are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$ it must

⁵ If there are only two edges of the path the same proof holds with $b = v$

Algorithm 5 MAIN-APASP(G, k').

create a set S with prob. $n^{-\beta}$ using Lemma 16;
 create an augmented hierarchy $\hat{S}_{\log n}^{n^{-\beta/\log n}}$ with S as the augmenting set;
 create a mixed hierarchy $\bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}}$ with $\bar{H}_1 = S$;
 $M_1 \leftarrow \text{AUG-APASP}(G, \hat{S}_{\log n}^{n^{-\beta/\log n}})$;
 $M_2 \leftarrow \text{MIX-APASP}(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$;
 $M \leftarrow \min\{M_1, M_2\}$;
 return M ;

be that either $a \notin B_{\bar{S}_1}(u)$ or $b \notin B_{\bar{S}_1}(v)$. Assume, wlog, that $a \notin B_{\bar{S}_1}(u)$. From the definition of r it must be that $u \in B_{\bar{S}_1}(a)$, as otherwise $r > 0$. Because $u \in B_{\bar{S}_1}(a)$ we update $\mathbf{w}(p_1(u), b)$ in the special phase of MIX-PIVOT-DIST so that it is at most $d_G(p_1(u), u) + d_G(u, a) + \mathbf{w}(a, b)$.

Since $r' = 0$ we have that $P(v, b) \in E_{\bar{S}_2}$. Thus, at the third phase of MIX-PIVOT-DIST we update $M[p_1(u), v]$, due to the bound of $\mathbf{w}(p_1(u), b)$, so that it is at most $d_G(p_1(u), u) + d_G(u, v)$. Now in line (1) of MIX-APSP $M[u, v]$ is updated so that it is at most $2d_G(p_1(u), u) + d_G(u, v)$. Since $a \notin B_{\bar{S}_1}(u)$ we have $d_G(p_1(u), u) \leq d_G(u, a)$ and we get that $M[u, v] \leq 2d_G(u, a) + d_G(u, v)$. This is exactly bound (i) in Claim 13 for $k = 2$ and thus holds for every $k \geq 2$. Thus, we can assume that $r > 0$ and not computing data for \bar{S}_0 in phase 3 of MIX-PIVOT-DIST has no affect on the proof of Claim 13 since $f \geq r$. \blacktriangleleft

4.3 Combining augmented hierarchies and mixed hierarchies

We now describe our main APASP algorithm MAIN-APASP. The input is a weighted undirected graph G and an integer k' . We use two hierarchies. An augmented hierarchy \hat{S}_k^p , $k = \log n$ levels, set \hat{S}_0 is V and the probability p is $\tilde{O}(n^{-\beta/\log n})$. The hierarchy is augmented with a set S computed by Lemma 16 with parameter $n^{-\beta}$. Notice that $|\hat{S}_{\log n}| = \tilde{O}(n^{1-\beta})$, and for every $v \in V$ we have $|B_{\hat{S}_{\log n}}(v)| = |C_{\hat{S}_{\log n}}(v)| = O(n^\beta)$. The second hierarchy is a mix hierarchy $\bar{\mathcal{H}}_{k'}^{p,q}$ with k' levels, $\bar{H}_1 = \hat{S}_{\log n}$ and $\bar{H}_{k'} = \emptyset$. Since $\hat{S}_{\log n}$ is formed using Lemma 16 with parameter $n^{-\beta}$ we have $q = n^{-\beta}$. The probability p is $\tilde{O}(n^{-\gamma/(k'-2)})$.

Next, we run $\text{AUG-APASP}(G, \hat{S}_{\log n}^{n^{-\beta/\log n}})$. Then we run $\text{MIX-APASP}(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$. MAIN-APASP is presented in Algorithm 5. We now turn to analyse the running time.

► **Lemma 25.** MAIN-APASP(G, k') runs in of $\tilde{O}(n^2 + m^{\frac{2}{k'}} n^{2-\frac{3}{k'}})$ expected time.

Proof. It follows from Lemma 16 that S can be computed in $\tilde{O}(mn^\beta)$ expected running time. It follows from Corollary 18 that the expected running time of $\text{AUG-APASP}(G, \hat{S}_{\log n}^{n^{-\beta/\log n}})$ is $\tilde{O}(n^2 + n^2 \cdot n^{\beta/\log n} \log n + mn^\beta \log n) = \tilde{O}(n^2 + mn^\beta)$. It follows from Corollary 23 that the expected running time of $\text{MIX-APASP}(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$ is $\tilde{O}(n^2 + (k' - 2)n^2 n^{\gamma/(k'-2)} + mn^{1-\beta-\gamma})$. We first express n^β as a function of n^γ using the equation $n^2 n^{\gamma/(k'-2)} = mn^{1-\beta-\gamma}$ which balances the two main terms in the running time of MIX-APASP.

$$n^2 n^{\gamma/(k'-2)} = mn^{1-\beta-\gamma} \longrightarrow n^\beta = \frac{mn^{1-\gamma}}{n^2 n^{\gamma/(k'-2)}} \longrightarrow n^\beta = \frac{m}{n^{1+\gamma(1+1/(k'-2))}}$$

Next, we substitute n^β as a function of n^γ in the running time of AUG-APASP.

$$\tilde{O}(n^2 + mn^\beta) \xrightarrow{n^\beta = \frac{m}{n^{1+\gamma(1+1/(k'-2))}}} \tilde{O}(n^2 + \frac{m^2}{n^{1+\gamma(1+1/(k'-2))}})$$

We now compute the value of n^γ using the equation $n^{2+\gamma/(k'-2)} = \frac{m^2}{n^{1+\gamma(1+1/(k'-2))}}$ which balances the two main terms in the running times of MIX-APASP and AUG-APASP.

$$n^{\gamma(1+2/(k'-2))} = \frac{m^2}{n^3} \longrightarrow n^\gamma = \left(\frac{m^2}{n^3}\right)^{\frac{k'-2}{k'}}$$

To obtain the running time of APASP we substitute n^γ in $n^{2+\gamma/(k'-2)}$:

$$n^{2+\gamma/(k'-2)} \xrightarrow{n^\gamma = \left(\frac{m^2}{n^3}\right)^{\frac{k'-2}{k'}}} n^2 \left(\frac{m^2}{n^3}\right)^{\frac{1}{k'}} = n^{2-\frac{3}{k'}} m^{\frac{2}{k'}}$$

Thus, we get that the running time of APASP is $\tilde{O}(n^2 + m^{\frac{2}{k'}} n^{2-\frac{3}{k'}})$. ◀

We now bound the approximation of MAIN-APASP.

► **Lemma 26.** *Algorithm MAIN-APASP(G, k') returns a matrix M that satisfies: $d_G(u, v) \leq M[u, v] \leq 2d_G(u, v) + \frac{k'-2}{k'} \cdot d_G(u, v)$, where $k' \geq 2$.*

Proof. Let $u, v \in V$. Recall that $\bar{H}_1 = \hat{S}_{\log n}$. If u and v are covered by $B_{\bar{H}_1}(u)$ and $B_{\bar{H}_1}(v)$ then it follows from Lemma 21 that Algorithm AUG-APASP returns a matrix M_1 that satisfies: $d_G(u, v) \leq M_1[u, v] \leq 2d_G(u, v)$. If u and v are not covered by $B_{\bar{H}_1}(u)$ and $B_{\bar{H}_1}(v)$ then it follows from Lemma 24 that Algorithm MIX-APASP returns a matrix M that satisfies:

$$d_G(u, v) \leq M_2[u, v] \leq 2d_G(u, v) + \frac{k'-2}{k'} \cdot d_G(u, v).$$

Since $M = \min\{M_1, M_2\}$ the claim follows. ◀

Proof of Theorem 1

Proof. The proof follows from Lemma 25 and Lemma 26. ◀

References

- 1 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.
- 2 Maor Akav and Liam Roditty. An almost 2-approximation for all-pairs of shortest paths in subquadratic time. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1–11. SIAM, 2020.
- 3 S. Baswana, V. Goyal, and S. Sen. All-pairs nearly 2-approximate shortest paths in $o(n^2 \text{poly log } n)$ time. *Theor. Comput. Sci.*, 410(1):84–93, 2009.
- 4 S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, 2010.
- 5 Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *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*, pages 609–621, 2008.
- 6 P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. WADS*, pages 541–552, 2007.
- 7 Shiri Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663, 2014.

- 8 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10, 2015.
- 9 E. Cohen and U. Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001.
- 10 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.
- 11 M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.
- 12 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths and PRAM distance oracles in weighted graphs. *CoRR*, abs/1907.11422, 2019.
- 13 Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016.
- 14 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016.
- 15 Telikepalli Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. *Algorithmica*, 63(1-2):224–245, 2012.
- 16 Mathias Bæk Tejs Knudsen. Additive spanners and distance oracles in quadratic time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 64:1–64:12, 2017.
- 17 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.
- 18 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *FOCS*, pages 109–118, 2006.
- 19 S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.
- 20 Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, 34(6):1398–1431, 2005.
- 21 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *JCSS*, 51:400–403, 1995.
- 22 A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. FOCS*, pages 605–614, 1999.
- 23 Christian Sommer. All-Pairs Approximate Shortest Paths and Distance Oracle Preprocessing. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:13, 2016.
- 24 A. Stothers. On the complexity of matrix multiplication. *Ph.D. Thesis, U. Edinburgh*, 2010.
- 25 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001.
- 26 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 2005.
- 27 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014.
- 28 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.
- 29 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 202–208, 2012.
- 30 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

The Voronoi Diagram of Rotating Rays

With applications to Floodlight Illumination

Carlos Alegría ✉ 

Dipartimento di Ingegneria, Università Roma Tre, Rome, Italy

Ioannis Mantas ✉ 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Evanthia Papadopoulou ✉ 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Marko Savić ✉ 

Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia

Hendrik Schrezenmaier ✉

Institut für Mathematik, Technische Universität Berlin, Germany

Carlos Seara ✉

Departament de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain

Martin Suderland ✉ 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Abstract

We introduce the *Voronoi Diagram of Rotating Rays*, a Voronoi structure where the input sites are rays, and the distance function is the counterclockwise angular distance between a point and a ray-site. This novel Voronoi diagram is motivated by illumination and coverage problems, where a domain has to be covered by floodlights (wedges) of uniform angle, and the goal is to find the minimum angle necessary to cover the domain. We study the diagram in the plane, and we present structural properties, combinatorial complexity bounds, and a construction algorithm. If the rays are induced by a convex polygon, we show how to construct the ray Voronoi diagram within this polygon in linear time. Using this information, we can find in optimal linear time the *Brocard angle*, the minimum angle required to illuminate a convex polygon with floodlights of uniform angle. This last algorithm improves upon previous results, settling an interesting open problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases rotating rays, Voronoi diagram, oriented angular distance, Brocard angle, floodlight illumination, coverage problems, art gallery problems

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.5

Funding C.A. was supported by MIUR Proj. “AHeAD” n° 20174LF3T8. Early work of I.M. and E.P. were supported by the DACH project Voronoi++, SNF-200021E_154387. M.S. is partly supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia (Grant No. 451-03-9/2021-14/200125), and the Provincial Secretariat for Higher Education and Scientific Research, Province of Vojvodina. Early work of H.S. was partly supported by the DFG grant FE-340/11-1. C.S. was supported by project PID2019-104129GB-I00/AEI/10.13039/501100011033. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.

Acknowledgements Initial discussions took place at the Intensive Research Program in Discrete, Combinatorial and Computational Geometry in Barcelona, Spain, in 2018. We are grateful to the organizers for providing the platform to meet and collaborate.



© Carlos Alegría, Ioannis Mantas, Evanthia Papadopoulou, Marko Savić, Hendrik Schrezenmaier, Carlos Seara, and Martin Suderland;

licensed under Creative Commons License CC-BY 4.0

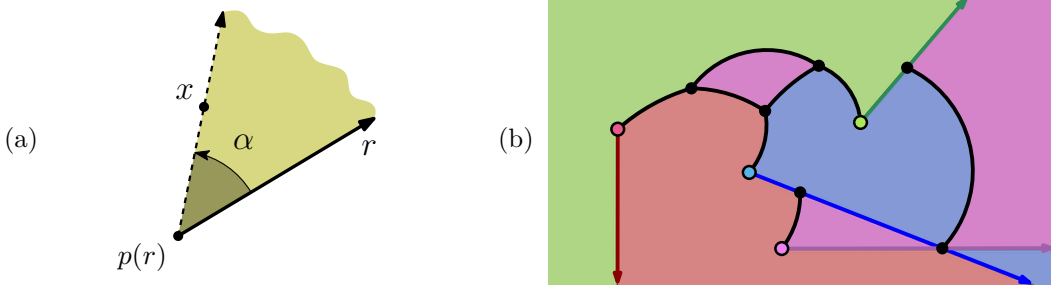
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 5; pp. 5:1–5:16



Leibniz International Proceedings in Informatics

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



■ **Figure 1** (a) An α -floodlight aligned with a ray r , having apex $p(r)$. The angle α is the angular distance from a point x to r . (b) The Rotating Rays Voronoi diagram of four rays in the plane. Each region is represented by a different color, and all points in a region are first illuminated by the ray of the respective color.

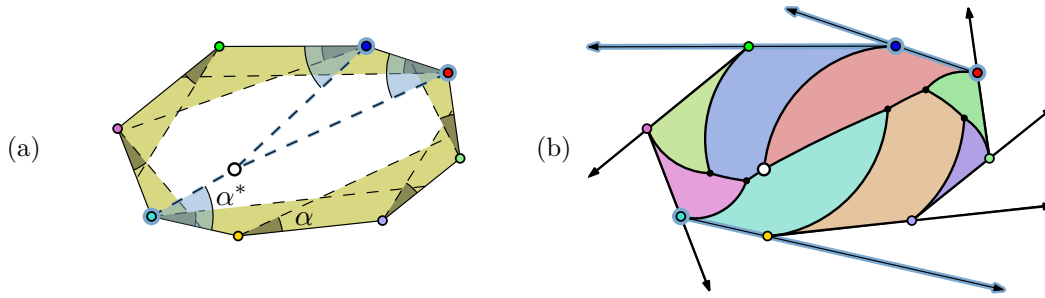
1 Introduction

In this work, we study the *Voronoi diagram of rotating rays* in the plane, which is defined by a set of n rays in \mathbb{R}^2 , under an angular distance function. Given a point x and a ray r in the plane, the *angular distance* from x to r is the smallest angle α such that, if we counterclockwise rotate r around its apex by α , it reaches (*illuminates*) x ; see Figure 1. We define this diagram and use it to solve related floodlight illumination problems.

Motivation. Illumination problems are well known *art gallery* type of problems, where a given domain has to be covered by so-called *floodlights*, which are light sources that illuminate a cone from their apex. The distance measure we study in this work is motivated by the following illumination problem. An α -floodlight is a floodlight with aperture α . Given a simple polygon P , an α -floodlight facing the interior of P is placed on each vertex v , in such a way that one of its rays contains the successor of v in the counterclockwise order of the vertices of P ; see Figure 2(a). The *Brocard Illumination problem* [1] asks for the *Brocard angle*, the smallest value of α for which the set of α -floodlights covers the interior of P .

When P is a convex polygon, the Brocard angle α^* can be revealed by constructing the Voronoi diagram of the rays placed at the vertices of P . In particular, α^* is realized at a vertex of the diagram with the maximum angular distance; see Figure 2(b). A rather natural extension is to follow a similar approach, not only in polygonal, but also in different domains. In addition to polygons, traditional domains to illuminate by floodlights include the entire plane, a region in the plane, or a single curve; see e.g., [6, 10, 25, 27]. Constructing the respective Voronoi diagram restricted to each domain yields the minimum angle needed by the floodlights to illuminate this domain. Hence, there is an interest in studying such Voronoi diagrams and in designing efficient construction algorithms for different domains.

Related work. In the Brocard illumination problem, a polygon P is called a *Brocard polygon*, if the Brocard angle is realized at a point, which is illuminated by all the floodlights, and is also equidistant to all the rays containing the sides of P [4]. The characterization of Brocard polygons has a long history, yet, only harmonic polygons (which include triangles and regular polygons) are known to be Brocard [7]. Nevertheless, we can detect if a convex polygon with n vertices is Brocard in $O(n)$ time, and compute the Brocard angle in such case in $O(1)$



■ **Figure 2** A polygon P . (a) Illumination by α -floodlights aligned with each edge of P . (b) The Voronoi diagram of the edge-aligned rays confined into P . Highlighted are the rays that realize the Brocard angle α^* , along with the point at which α^* is realized.

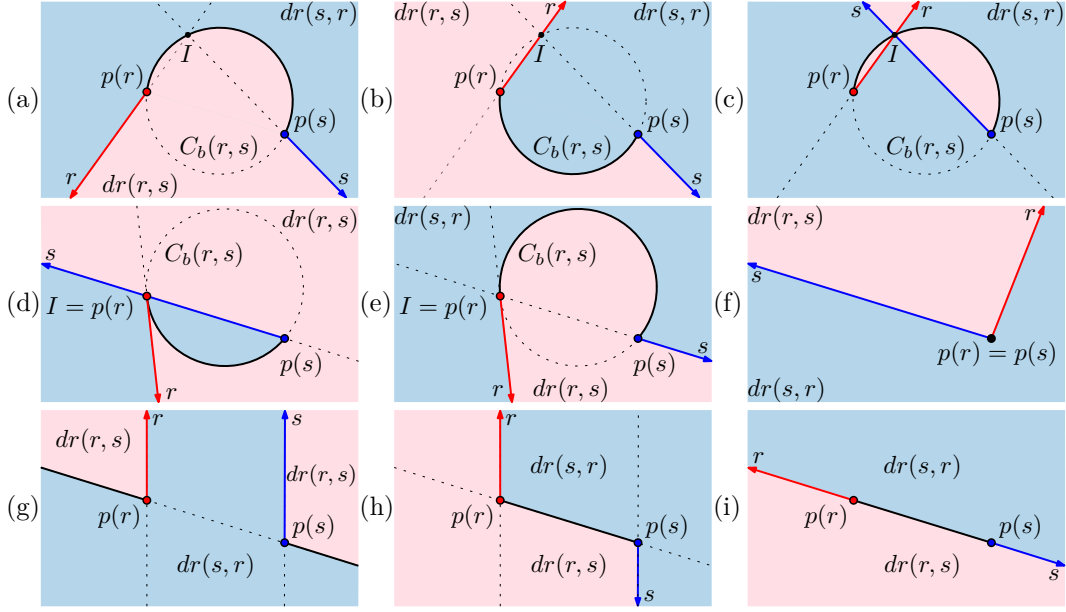
time. Algorithms for the computation of the Brocard angle of arbitrary simple polygons were recently presented by Alegría et al. [1]. The authors gave an $O(n^3 \log^2 n)$ time algorithm and complemented this with an $O(n \log n)$ -time algorithm for convex polygons¹.

Since their introduction [6], floodlight illumination problems have been studied a lot, see e.g., [22, 28]. The case when the floodlights are of uniform angle has also been explored by several authors, see e.g., [8, 13, 16, 23, 27]. From a different viewpoint, rotating α -floodlights can also be used to model devices with limited sensing range, like surveillance cameras or directional antennae; see [3, 9, 20, 21]. In this context, the minimum angle is interpreted as the minimum range needed by a set of devices to cover a domain. The Rotating Rays Voronoi Diagram is novel with respect to both the input sites and the distance function. A somewhat related diagram was defined in [11] to model dominance regions in the analysis of soccer matches [26].

Our contribution. We introduce the *Rotating Rays Voronoi Diagram* and prove a series of results, paving the way for future work on similar problems. Our interest in this diagram stems from its application to floodlight illumination in different domains.

- In Section 3, we consider the diagram in the plane. We identify structural properties which we complement with complexity results: an $\Omega(n^2)$ worst case lower bound and an $O(n^{2+\epsilon})$ upper bound. As a by-product we get an $O(n^{2+\epsilon})$ -time construction algorithm. This algorithm allows us to find, within the same time, the minimum angle needed to illuminate the plane with a given set of ray-aligned floodlights.
- Motivated by the Brocard illumination problem, in Section 4, we restrict our domain to a convex polygonal region bounded by the input set of sites. We study the Voronoi diagram in such domain and describe a construction algorithm that works in deterministic $\Theta(n)$ time. As our main contribution, we show how to use the resulting diagram to find the Brocard angle of a convex polygon in optimal linear time.
- Finally, in Section 5, we consider the case where the domain of interest is a curve, and we show how to construct the diagram along various types of curves.

¹ The $O(n)$ time analysis of the algorithm for convex polygons stated in [1] is not correct.



■ **Figure 3** The bisector of two rays r and s , consisting of r (red ray), s (blue ray) and an arc of $C_b(r, s)$ (black curve), in different configurations. (a) Non-intersecting with $I \notin r \cup s$. (b) Non-intersecting with $I \in r$. (c) Intersecting. (d) “Tangent” with $p(r) \in s$. (e) “Tangent” with $p(r) \in l(s) \setminus s$. (f) Sharing their apex. (g) Parallel. (h) Anti-parallel with $l(r) \neq l(s)$. (i) Anti-parallel with $l(r) = l(s)$.

2 Preliminaries

Let S be a set of n rays in \mathbb{R}^2 . Given a ray r , we denote its apex by $p(r)$, its supporting line by $l(r)$, and its direction in the unit circle by $\hat{d}(r)$. We define the distance function as follows.

► **Definition 1.** Given a ray r and a point $x \in \mathbb{R}^2$, the oriented angular distance from x to r , denoted by $d_{\angle}(x, r)$, is the minimum counterclockwise angle α from r to a ray with apex $p(r)$ passing through x ; see Figure 1(a). Further, we define $d_{\angle}(p(r), r) = 0$.

It is easy to see that the oriented angular distance is not a metric. Moreover, observe that $d_{\angle}(x, r) \in [0, 2\pi)$, and there is a discontinuity at 2π . Using this distance function, we define the bisector of two rays and the Voronoi diagram of a set of rays.

► **Definition 2.** Given two rays r and s , the dominance region of r over s , denoted by $dr(r, s)$, is the set of points with smaller angular distance to r than to s , i.e., $dr(r, s) := \{x \in \mathbb{R}^2 \mid d_{\angle}(x, r) < d_{\angle}(x, s)\}$. The angular bisector of r and s , denoted by $b_{\angle}(r, s)$, is the curve delimiting $dr(r, s)$ and $dr(s, r)$; see Figure 3.

Note that, due to the discontinuity of the distance function, our definition of a bisector is slightly different from standard, which is the locus of points equidistant from two sites.

Given two rays r and s , let $I = l(r) \cap l(s)$. The bisector $b_{\angle}(r, s)$ is the union of the two rays r and s , and a circular arc a that connects $p(r)$ to $p(s)$; see Figure 3. The arc a belongs to the bisecting circle $C_b(r, s)$, which we define as follows:

- If I , $p(r)$, and $p(s)$ are three distinct points, then $C_b(r, s)$ is the circle through I , $p(r)$, and $p(s)$. The arc a contains I , if and only if I lies either on none or on both of r and s ; see Figure 3(a), Figure 3(b) and Figure 3(c).

- If $I = p(r)$ and $I \neq p(s)$, then $C_b(r, s)$ is the circle tangent to $l(r)$ passing through $p(r)$ and $p(s)$. Both a and r lie on the same side of $l(s)$, if and only if $p(r)$ lies on s . We analogously define $C_b(r, s)$ if $I = p(s)$ and $I \neq p(r)$; see Figure 3(d) and Figure 3(e).
- If $p(r) = p(s)$, then both $C_b(r, s)$ and a degenerate to a single point; see Figure 3(f).
- If $l(r)$ and $l(s)$ are parallel, then $C_b(r, s)$ degenerates to the line through $p(r)$ and $p(s)$. If $\hat{d}(r) = -\hat{d}(s)$, then a is a line segment. If instead $\hat{d}(r) = \hat{d}(s)$, then a consists of two half-lines; see Figure 3(g), Figure 3(h) and Figure 3(i).

Observe that the oriented angular distance is monotone along the circular arc a . Further, the angular bisector is connected, unless its defining rays are parallel. For the sake of simplicity, unless otherwise stated, we assume that no two rays in \mathcal{S} have parallel supporting lines, and that the apex of any ray does not lie on any other ray.

► **Definition 3.** *The Rotating Rays Voronoi Diagram (RVD) of a set of rays \mathcal{S} is the subdivision of \mathbb{R}^2 into Voronoi regions defined as follows:*

$$vreg(r) := \{x \in \mathbb{R}^2 \mid \forall s \in \mathcal{S} \setminus \{r\} : d_{\angle}(x, r) < d_{\angle}(x, s)\}.$$

The graph structure of the diagram is $RVD(\mathcal{S}) := (\mathbb{R}^2 \setminus \bigcup_{r \in \mathcal{S}} vreg(r)) \cup \mathcal{S}$.

The Voronoi region $vreg(r)$ can be equivalently defined as the intersection of all the dominance regions of r , i.e., $vreg(r) = \bigcap_{s \in \mathcal{S} \setminus \{r\}} dr(r, s)$. A Voronoi region may be disconnected; each connected component of a Voronoi region is called a *face*.

The diagram $RVD(\mathcal{S})$ has different types of edges and vertices. An edge can be a line segment, a half-line, or a circular arc. A vertex can be the apex of a ray, the intersection point of two rays, the intersection point of three circular arcs, or the intersection point of a ray and a circular arc. Further, $RVD(\mathcal{S})$ is a planar graph with bounded maximum degree; thus, to bound its complexity, it suffices to bound any of its edges, vertices, or faces.

3 RVD in the Plane

In this section we study the diagram $RVD(\mathcal{S})$ in the plane. We first look at some properties and combinatorial complexity bounds. Then we consider the problem of illuminating the plane with a set of floodlights aligned with \mathcal{S} .

3.1 Properties, complexity, and an algorithm

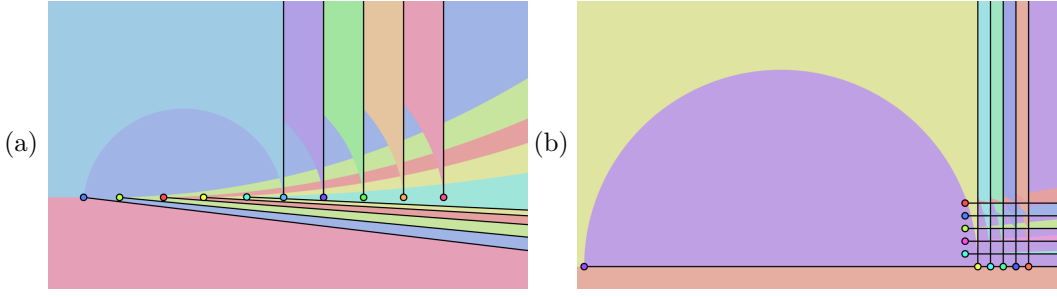
The diagram $RVD(\mathcal{S})$ satisfies the following two simple structural properties.

► **Lemma 4.** *$RVD(\mathcal{S})$ has exactly n unbounded faces, one for each ray in \mathcal{S} .*

Proof. Let r be an arbitrary ray of \mathcal{S} . To examine the unbounded portion of $vreg(r)$, consider the intersection of $RVD(\mathcal{S})$ with a circle Γ of sufficiently large radius, enclosing the vertices of $RVD(\mathcal{S})$ and the bisecting circles of all the bisectors of \mathcal{S} .

For $s \in \mathcal{S} \setminus \{r\}$, the intersection $dr(r, s) \cap \Gamma$ is a circular arc of Γ going counterclockwise from $r \cap \Gamma$ to $s \cap \Gamma$. The region $vreg(r)$ is the intersection of all the dominance regions of r , thus, $vreg(r) \cap \Gamma$ is the intersection of $n - 1$ circular arcs, all starting from r . Hence, $vreg(r) \cap \Gamma$ is a single arc starting from r and ending at the first ray on the counterclockwise ordering along Γ . So, for any ray r , the region $vreg(r)$ has exactly one unbounded face. ◀

► **Lemma 5.** *$RVD(\mathcal{S})$ is connected.*



■ **Figure 4** (a) A set of pairwise non-intersecting rays with $\text{RVD}(\mathcal{S})$ having $\Theta(n^2)$ complexity. (b) A diagram with a Voronoi region (purple ray - leftmost apex) having $\Theta(n^2)$ complexity.

Proof. Assume that $\text{RVD}(\mathcal{S})$ is not connected, hence, it has at least two connected components. Then some region $\text{vreg}(r)$ disconnects $\text{RVD}(\mathcal{S})$ by either having an unbounded face with two occurrences at infinity, or by enclosing a component creating conceptually an *island*.

The first case is excluded by the proof of Lemma 4. For the second case, consider the disconnected component of $\text{RVD}(\mathcal{S})$ surrounded by $\text{vreg}(r)$. This component consists of at least one face of a region $\text{vreg}(s)$ for some $s \in \mathcal{S}$. Then, in $\text{RVD}(\{r, s\})$, there is an island inside $\text{vreg}(r)$. Thus, $b_{\angle}(r, s)$ has a bounded connected component, in contradiction to the fact that each bisector is a single unbounded curve. ◀

We now study the combinatorial complexity of $\text{RVD}(\mathcal{S})$. An $\Omega(n^2)$ lower bound is easily achieved by a set \mathcal{S} of n pairwise intersecting rays. In such case, $\text{RVD}(\mathcal{S})$ has $\binom{n}{2} = \Theta(n^2)$ vertices (one per intersection of rays) and thus $\Omega(n^2)$ complexity. Next, we show that this bound also holds for pairwise non-intersecting rays.

► **Theorem 6.** *The worst case combinatorial complexity of $\text{RVD}(\mathcal{S})$ has an $\Omega(n^2)$ lower bound, even if the rays are pairwise non-intersecting.*

Proof. We give the following construction illustrated in Figure 4(a), where the Voronoi regions of the $n/2 - 1$ rays with the leftmost apices have $n/2 + 1$ faces each. We set $n = 2m$ and let $p(r_i) = (i, 0)$, $i = 1, \dots, 2m$, with rays r_{m+1}, \dots, r_{2m} pointing vertically upwards. For $i = 1, \dots, m$, let the direction of r_i be $\hat{d}(r_i) = (\sin \alpha_i, \cos \alpha_i)$ with $\alpha_1 \in (3\pi/2, 2\pi)$ and $\alpha_i = \alpha_{i-1} + \epsilon_i$ where $\epsilon_i > 0$ for $i = 2, \dots, m$. We choose ϵ_i one by one, in the increasing order of i , so that both r_i and r_{i+1} have a face between any two consecutive upward shooting rays. This is always possible since we can choose ϵ_i small enough so that, at any x -coordinate $x < 2m$, the circular arc of $b_{\angle}(r_i, r_{i+1})$ is arbitrarily close to the x -axis and, thus, it is below the circular arc of $b_{\angle}(r_{i-1}, r_i)$. ◀

► **Theorem 7.** *A Voronoi region of $\text{RVD}(\mathcal{S})$ has $\Theta(n^2)$ complexity in the worst case.*

Proof. We first argue that the complexity of a region is $O(n^2)$. A vertex v of $\text{RVD}(\mathcal{S})$ can be defined by a triplet of rays $r, s, t \in \mathcal{S}$. The bisectors $b_{\angle}(r, s)$ and $b_{\angle}(r, t)$ intersect $O(1)$ times, hence, $\text{RVD}(\{r, s, t\})$ has $O(1)$ vertices. Now consider a ray r and its region $\text{vreg}(r)$. All but at most $O(n)$ vertices on the boundary of $\text{vreg}(r)$ are defined by r and a pair of sites. There are $\Theta(n^2)$ pairs, each inducing $O(1)$ vertices on $\text{vreg}(r)$, so $\text{vreg}(r)$ has $O(n^2)$ vertices.

We now give a construction of $n = 2m + 1$ rays, where a single region has $\Theta(n^2)$ complexity; refer to the construction of Figure 4(b). We first create a grid structure. For $i = 1, \dots, m$, let r_i be a ray with $p(r_i) = (i, 0)$ shooting vertically upward, and let s_i be a ray with $p(s_i) = (0, i)$ shooting horizontally to the right. For all $(i, j) \in \{1, \dots, m - 1\}^2$,

let $R(i, j)$ be the square $[i, i + 1) \times [j, j + 1)$. Each square $R(i, j)$ is made up of two faces of $\text{RVD}(\{r_1, \dots, r_m, s_1, \dots, s_m\})$, one belonging to $\text{vreg}(r_i)$ and one belonging to $\text{vreg}(s_j)$. Now let $\alpha(i, j) := \max\{\min\{d_{\angle}(x, r_i), d_{\angle}(x, s_j)\} \mid x \in R(i, j)\}$ and let $\alpha_{\min} := \min\{\alpha(i, j) \mid (i, j) \in \{1, \dots, m-1\}^2\}$. It is easy to see that $\alpha_{\min} < \arctan 1/(m-1)$.

We now introduce another ray t , so that $\max\{d_{\angle}(x, t) \mid x \in [1, n-1]^2\} < \alpha_{\min}$. This can be achieved if $p(t) = (-n^2, 0)$ and t is shooting horizontally to the right. This means that in each $R(i, j)$, for $(i, j) \in \{1, \dots, n-1\}^2$, t will visit some point before any of the rays r_i or s_j , implying that $\text{vreg}(t)$ has $\Theta(n^2)$ faces. ◀

We now show how the angular distance function can be adapted in order to apply the general upper bounds of Sharir [24]. As a by-product we also obtain an algorithm for $\text{RVD}(\mathcal{S})$.

► **Theorem 8.** *$\text{RVD}(\mathcal{S})$ has $O(n^{2+\epsilon})$ combinatorial complexity, for any $\epsilon > 0$. Further, $\text{RVD}(\mathcal{S})$ can be constructed in $O(n^{2+\epsilon})$ time.*

Proof. Each site r induces a function $d_{\angle}^r(x) = d_{\angle}(x, r)$ which maps a point $x = (x_1, x_2) \in \mathbb{R}^2$ to its angular distance from r . The RVD can be seen as the projection of the lower envelope of the graphs of these distance functions in 3-space to the plane. For algebraic distance functions, Sharir [24] gives complexity bounds for this lower envelope accompanied with algorithmic results. The angular distance functions though are not algebraic. Our strategy is to find algebraic functions d_{alg}^r that are equivalent to the functions d_{\angle}^r for the computation of the lower envelope, i.e., they fulfill the following property for all $r, s \in \mathcal{S}$ and $x \in \mathbb{R}^2$:

$$d_{\angle}^r(x) < d_{\angle}^s(x) \Leftrightarrow d_{\text{alg}}^r(x) < d_{\text{alg}}^s(x).$$

Without loss of generality, assume that $p(r)$ lies on the origin and r is facing to the right, that is, in the positive x_1 -direction of the coordinate system. Let $x \in \mathbb{R}^2$ and $\alpha := d_{\angle}^r(x)$. Then we want to set $d_{\text{alg}}^r(x) := 1 - \cos(\alpha)$ if $0 \leq \alpha \leq \pi$, and $d_{\text{alg}}^r(x) := 3 + \cos(\alpha)$ if $\pi < \alpha < 2\pi$. The function $x \mapsto \cos(\alpha)$ is indeed algebraic since it is obtained by first scaling x to unit length and then mapping it to its first coordinate. Then we have

$$d_{\text{alg}}^r(x) = \begin{cases} 0 & \text{if } x_1 = x_2 = 0, \\ 1 - \frac{x_1}{\sqrt{x_1^2 + x_2^2}} & \text{if } x_1 \neq 0, x_2 \geq 0, \\ 3 + \frac{x_1}{\sqrt{x_1^2 + x_2^2}} & \text{otherwise.} \end{cases}$$

Since d_{alg}^r consists of three patches, which are all algebraic and have simple domain boundaries, applying [24] to these functions yields the claimed results. ◀

3.2 Minimum angle needed to illuminate the plane

Given a set \mathcal{S} of n rays in \mathbb{R}^2 , the Brocard Illumination problem can be naturally extended to the plane as follows. An α -floodlight f is *aligned with* a ray r , if f is equal to r when $\alpha = 0$. Let f_r denote a floodlight aligned with the ray $r \in \mathcal{S}$. The goal is to find the minimum angle α^* for which the set of α^* -floodlights $\{f_r \mid r \in \mathcal{S}\}$ illuminates the entire plane.

The angle α^* is realized at a point x^* that has the maximum angular distance to its nearest ray, that is, $\alpha^* = \max_{x \in \mathbb{R}^2} \min_{r \in \mathcal{S}} d_{\angle}(x, r)$. Hence, the point x^* necessarily lies on $\text{RVD}(\mathcal{S})$. Since the distance along a circular edge is monotone, it follows that x^* cannot lie on such an edge. Thus, x^* is either a vertex of $\text{RVD}(\mathcal{S})$ or a point at infinity on a ray of \mathcal{S} . We can find x^* , and hence α^* , by first constructing $\text{RVD}(\mathcal{S})$ in $O(n^{2+\epsilon})$ time, and then traversing the diagram in linear time in its size. We obtain the following result.

► **Theorem 9.** *Given a set of rays \mathcal{S} and an α -floodlight aligned with each ray, the minimum angle α^* needed to illuminate \mathbb{R}^2 can be found in $O(n^{2+\epsilon})$ time, for any $\epsilon > 0$.*

Next, we give tight bounds on the value of α^* .

► **Proposition 10.** *Given a set of rays \mathcal{S} and an α -floodlight aligned with each ray, the angle α^* is greater or equal to $2\pi/n$. Further, α^* can take any value in the interval $[2\pi/n, 2\pi)$.*

Proof. For the lower bound, consider that in order to illuminate the entire plane, all the points at infinity should also be illuminated. To illuminate these points, the sum of the angles of all rays, should be at least 2π . Hence, in the best case, each point at infinity is seen by exactly one ray, and the $2\pi/n$ lower bound follows. Next, we give a construction realizing this bound.

Let \mathcal{S} be a set of rays having their apices on $(0,0)$, with the property that any two consecutive rays have an angular difference of $2\pi/n$. The last points to be illuminated will be all the points on the right side of each ray r_i . These points are illuminated simultaneously by r_{i-1} when α reaches $2\pi/n$. Further, this construction can be easily adapted to attain any value in $(2\pi/n, 2\pi)$, by expanding a wedge bounded by two consecutive rays to the desired angle and shrinking all the other wedges analogously. ◀

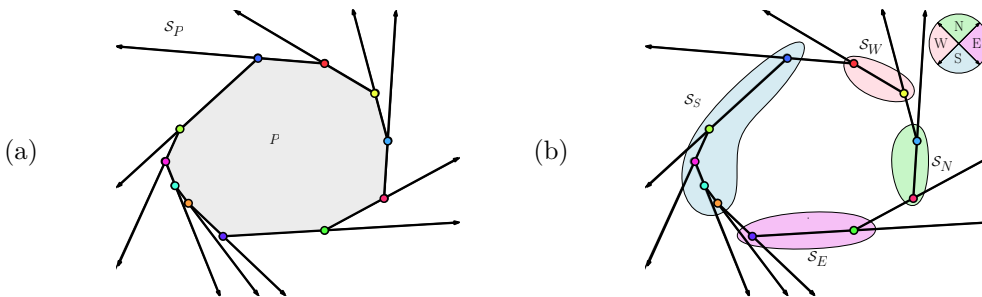
4 RVD of a convex polygon: Brocard illumination

We now turn our attention to the Brocard illumination problem. We are given a convex polygon P with n vertices, and we want to find the Brocard angle α^* of P . By computing an RVD restricted to P , we show how to compute α^* in optimal $\Theta(n)$ time.

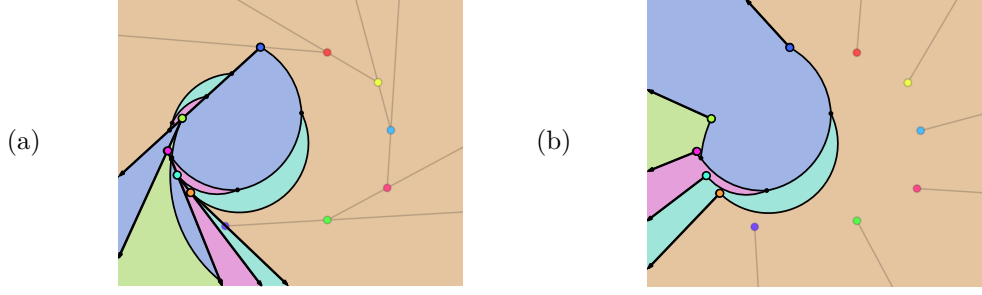
Let \mathcal{S}_P be the set of n rays such that each ray has a vertex $v \in P$ as apex, and passes through the successor vertex of v in a counterclockwise traversal of P ; see Figure 5(a). Let $\text{PRVD}(\mathcal{S}_P)$ be the diagram $\text{RVD}(\mathcal{S}_P)$ restricted to the interior of P . We first describe an algorithm to construct $\text{PRVD}(\mathcal{S}_P)$ and then show to use this diagram to obtain α^* . Note that we do not construct the complete $\text{RVD}(\mathcal{S}_P)$, which may have $\Theta(n^2)$ complexity, but only $\text{PRVD}(\mathcal{S}_P)$, which has $\Theta(n)$ size.

4.1 Algorithm to compute RVD of a convex polygon

The general strategy is to split the problem into four subproblems. Each subproblem will satisfy a set of conditions that allows us to use the existing $\Theta(n)$ time algorithms, which are based on *abstract Voronoi diagrams* [17, 19]. We then complete our construction by merging the resulting diagrams.



■ **Figure 5** (a) A convex polygon P and the corresponding set of rays \mathcal{S}_P . (b) The partitioning of \mathcal{S}_P into the four subsets $\mathcal{S}_N, \mathcal{S}_W, \mathcal{S}_S$ and \mathcal{S}_E , depending on the direction of the rays.



■ **Figure 6** (a) The diagram $\text{RVD}(\mathcal{S}_S)$; some Voronoi regions are disconnected. (b) The diagram $\text{RVD}(\mathcal{S}_S^r)$, after a clockwise rotation of $\pi/2$; all Voronoi regions are connected.

More specifically, we first partition \mathcal{S}_P into four sets $\mathcal{S}_N, \mathcal{S}_W, \mathcal{S}_S$ and \mathcal{S}_E depending whether a ray points north, west, south or east respectively; see Figure 5(b). In this way, rays in a subset are consecutive and the direction of any two rays have a difference of at most $\pi/2$. For each set \mathcal{S}_d , $d \in \{N, W, S, E\}$, we obtain a set \mathcal{S}_d^r in which every ray of \mathcal{S}_d is rotated clockwise by an angle of $\pi/2$; see Figure 6. Then, we construct each diagram $\text{RVD}(\mathcal{S}_d^r)$ independently. Finally, we merge the four diagrams to obtain $\text{PRVD}(\mathcal{S}_P)$; see Figure 8.

4.2 The four diagrams of \mathcal{S}_d^r

To construct the diagram of each subset \mathcal{S}_d^r in $\Theta(|\mathcal{S}_d^r|)$ time, we make use of the *abstract Voronoi diagrams* framework [17, 18]. To fall under this framework, the system of angular bisectors must satisfy the following three *axioms*:

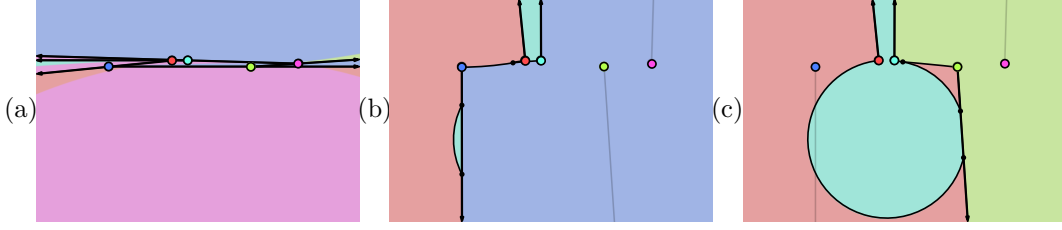
- **(A1)** The bisector $b_{\angle}(r, s)$, $\forall r, s \in \mathcal{S}$, is an unbounded Jordan curve.
- **(A2)** The region $\text{vreg}(r)$ in $\text{RVD}(\mathcal{S}')$, $\forall \mathcal{S}' \subseteq \mathcal{S}$ and $\forall r \in \mathcal{S}'$, is non-empty and connected.
- **(A3)** The closure of the union of all regions in $\text{RVD}(\mathcal{S}')$, $\forall \mathcal{S}' \subseteq \mathcal{S}$, covers \mathbb{R}^2 .

► **Lemma 11.** *The system of bisectors of \mathcal{S}_d^r satisfies the axioms (A1)-(A3).*

Proof sketch. **(A1):** Consider a pair of rays $r, s \in \mathcal{S}_d^r$. Let ℓ be the line through $p(r)$ and $p(s)$. Let $x \neq p(r)$ and $y \neq p(s)$ denote respectively, two points lying on r and s . Observe that the angles $\angle(p(s), p(r), x)$, $\angle(x, p(r), p(s))$, $\angle(y, p(s), p(r))$, and $\angle(p(r), p(s), y)$ are all greater than or equal to $\frac{\pi}{2}$, where $\angle(x, y, z)$ denotes the counterclockwise angle at point y , between the rays with apex y passing through x and z , respectively. Hence, r and s are non-intersecting regardless of whether both are contained on ℓ , or lie on the same side of ℓ . The bisector of two non-intersecting rays is an unbounded Jordan curve.

(A2): A Voronoi region $\text{vreg}(r)$ is obviously non-empty as it contains the ray r . Regarding the region connectivity, we claim that no ray of \mathcal{S}_d^r cuts twice a bisecting circle induced by the rays of \mathcal{S}_d^r . From the analysis of Section 2, this implies that every region of $\text{RVD}(\mathcal{S}_d^r)$ is connected, since it consists of a single unbounded face. Consider three rays $r, s, t \in \mathcal{S}_d$. The angle $-\alpha$ required to rotate r around $p(r)$ such that $l(r)$ is either tangent to or not intersecting $C_b(s, t)$ is strictly greater than $-\frac{\pi}{2}$. After rotating r by $-\frac{\pi}{2} \in [-\alpha, -(\alpha + \pi)]$, we obtain a ray in \mathcal{S}_d^r that does not intersect twice $C_b(r, s)$; see Figure 6(b). Our claim follows since \mathcal{S}_d and \mathcal{S}_d^r induce the same set of bisecting circles.

(A3): The diagram $\text{RVD}(\mathcal{S}_d^r)$ is defined by distance functions, one for each site in \mathcal{S}_d^r , whose domain is the entire plane. Thus, it follows that any point in the plane belongs to the closure of a region of $\text{RVD}(\mathcal{S}_d^r)$. ◀



■ **Figure 7** (a) $\text{RVD}(\mathcal{S}_P)$ of a polygon with five vertices. (b) A subset of three rays rotated by $-\pi/2$. The teal region has two faces; for it to be connected, the rays should be rotated by an angle greater than $-\pi/2$. (c) A subset of three rays rotated by $-\pi/2$. The red region has two faces; for it to be connected, the rays should be rotated by an angle smaller than $-\pi/2$.

Since each Voronoi region is connected, and it has exactly one unbounded face, as shown in Lemma 4, it follows that $\text{RVD}(\mathcal{S}_d^r)$ is a tree of complexity $\Theta(\mathcal{S}_d^r)$.

It is worth to note that the original set \mathcal{S}_P need not satisfy axioms (A1)-(A3), thus, $\text{RVD}(\mathcal{S}_P)$ need not fall under the framework of abstract Voronoi diagrams, and hence, the partition into the four subsets and the rotation.

The intuition behind the rotation of the rays comes from the fact that only circular parts of bisectors appear in $\text{PRVD}(\mathcal{S}_P)$, and that the bisecting circles remain the same under a uniform rotation. The partitioning of \mathcal{S}_P into the specific four sets is justified by the following remark.

► **Remark 12.** There are sets of rays \mathcal{S}_P for which there exists no unique angle to rotate the rays in \mathcal{S}_P , so that (A2) is satisfied. Refer to Figure 7.

$\text{RVD}(\mathcal{S}_d^r)$ can be constructed in $O(n \log n)$ time [17]. We can construct $\text{RVD}(\mathcal{S}_d^r)$ in $O(n)$ time by showing that the system of bisectors of \mathcal{S}_d^r falls under the *Hamiltonian abstract Voronoi diagram* framework of Klein and Lingas [19]. In addition to satisfying (A1)-(A3), the *cyclic version* of [19] requires the following axiom to be satisfied:

- **(A4)** There exists a Jordan curve \mathcal{H} of constant complexity such that, \mathcal{H} visits the region $\text{vreg}(r)$ in $\text{RVD}(\mathcal{S}')$, $\forall \mathcal{S}' \subseteq \mathcal{S}$ and $\forall r \in \mathcal{S}'$, exactly once.

If a Voronoi diagram satisfies axioms (A1)-(A4) and the ordering of the regions of $\text{RVD}(\mathcal{S}')$ along \mathcal{H} is given, then $\text{RVD}(\mathcal{S})$ can be computed in $\Theta(n)$ -time [19]. Hence, it suffices to find a curve \mathcal{H} satisfying these properties. We show this in the following.

► **Lemma 13.** $\text{RVD}(\mathcal{S}_d^r)$ can be constructed in deterministic $\Theta(|\mathcal{S}_d|)$ time.

Proof. We show that \mathcal{S}_d^r satisfies axiom (A4) of the Hamiltonian abstract Voronoi diagram framework [19]. The linear time algorithm is then a direct corollary of the existing results [19].

Let Γ be a circle of sufficient large radius enclosing all bisecting circles. Γ is obviously a Jordan curve of constant complexity. For any $\mathcal{S}' \subseteq \mathcal{S}_d^r$, the diagram $\text{RVD}(\mathcal{S}')$ is a tree, and by definition, Γ does not intersect any bisecting circle, hence, Γ visits each region of $\text{RVD}(\mathcal{S}')$ exactly once, with a change on the visited region happening when Γ intersects a ray. Γ is a Hamiltonian curve \mathcal{H} satisfying axiom (A4).

The ordering of the unbounded faces of $\text{RVD}(\mathcal{S}_d^r)$ corresponds to the ordering of the respective vertices along the polygon P , and this is maintained for any $\mathcal{S}' \subset \mathcal{S}_d^r$. The ordering of the vertices of P is given, so this concludes the proof. ◀

4.3 Merging the four diagrams

We now merge all the diagrams to obtain $\text{PRVD}(\mathcal{S}_P)$. Our merging procedure consists of two steps; see Figure 8. In an initial step we merge $\text{RVD}(\mathcal{S}_W^r)$ with $\text{RVD}(\mathcal{S}_S^r)$ to obtain $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$, and analogously we obtain $\text{RVD}(\mathcal{S}_E^r \cup \mathcal{S}_N^r)$. Then, in a final step we merge the diagrams $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$ and $\text{RVD}(\mathcal{S}_E^r \cup \mathcal{S}_N^r)$, restricted to the interior of P , to obtain $\text{PRVD}(\mathcal{S}_P)$. We show the following statement.

► **Lemma 14.** *Given $\text{RVD}(\mathcal{S}_d)$, for all $d \in \{N, W, S, E\}$, $\text{PRVD}(\mathcal{S}_P)$ can be constructed in $\Theta(n)$ time.*

We now describe the two merging steps and sketch the arguments of the proof. Refer also to Figure 8 and Figure 9. Note that, although the four Voronoi diagrams $\text{RVD}(\mathcal{S}_d)$ to be merged fall under the abstract Voronoi framework, this is not the case for the resulting diagrams. This requires special care in the merging process.

Initial step. We describe how to merge $\text{RVD}(\mathcal{S}_W^r)$ with $\text{RVD}(\mathcal{S}_S^r)$. Let w_1, \dots, w_k be the rays in \mathcal{S}_W^r and let s_1, \dots, s_l be the rays in \mathcal{S}_S^r , as they appear ordered on P . We need to construct the *merge curve*, which consists of the rays s_1, w_1 and the set of circular edges of $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$ equidistant to sites $w \in \mathcal{S}_W^r$ and $s \in \mathcal{S}_S^r$, which we denote by E_C . We describe each procedure separately.

- **Tracing s_1 :** The ray s_1 lies entirely in $\text{vreg}(w_k)$. To see this, consider the set \mathcal{S}_W (before rotation) and continuously clockwise rotate all rays by an angle of $\pi/2$. During this process, w_k does not intersect any of the rays in \mathcal{S}_W , hence, $s_1 \in \text{vreg}(w_k)$. Thus, s_1 does not intersect $\text{RVD}(\mathcal{S}_W^r)$ and it can be trivially traced in $O(1)$ time.
- **Tracing E_C :** We start from the point $p(s_1)$, using the bisector $b_{\angle}(s_1, w_k)$ and we trace a chain of circular edges until we reach the ray w_1 . In order to identify at each step the next bisector to proceed, we adapt standard tracing techniques for Voronoi diagrams, see e.g., [2]. More specifically, while going from s_1 to w_1 , the curve E_C visits the regions of the sites s_1, \dots, s_l in exactly that order (some sites might possibly not appear at all). The same holds for the sites w_1, \dots, w_k but in reverse order. Hence, to trace E_C we can go over the two lists of regions in the aforementioned order.
- **Structure of E_C :** To see that the chain ends up at w_1 , consider the distance of any point on the chain to the nearest ray. The distance when the chain starts, at $p(s_1)$, is exactly $\pi/2$, and it is monotonically increasing. Moreover, consider the polygonal chain P^* consisting of the line segments $\overline{p(w_1)p(w_2)}, \overline{p(w_2)p(w_3)}, \dots, \overline{p(w_k)p(s_1)}, \overline{p(s_1)p(s_2)}, \dots, \overline{p(s_{l-1})p(s_l)}$ and the ray s_l . The distance of any point on P^* to its nearest ray, is exactly $\pi/2$. Hence, since the distance along the chain of circular edges is increasing, the chain may only end up at w_1 (but not necessarily at its apex, see e.g., Figure 9(a)).

We now need to show that the chain, which we traced, is the complete set E_C , i.e., there are no other connected components to identify. Additionally, we have to show that E_C does not induce any bounded faces in $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$. To prove both statements we define an auxiliary Voronoi diagram, where in contrast to Definition 2, we define the bisector between two rays r and s , to be the entire circle $C_b(r, s)$.

We use this auxiliary diagram due to the simplicity of its bisectors. Since only the circular arcs of the bisectors appear inside P , some properties of the auxiliary Voronoi diagram also hold for $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$. More specifically, each region in the auxiliary diagram is connected and the union of all the regions covers P . In addition, the auxiliary Voronoi diagram implies that $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$ has $\Theta(|\mathcal{S}_W^r| + |\mathcal{S}_S^r|)$ size, hence, E_C has $O(n)$ size. The curve E_C has $O(n)$ size, and there is no backtracking while traversing the two lists of regions (to identify the edges), hence, E_C can be constructed in time $O(n)$.

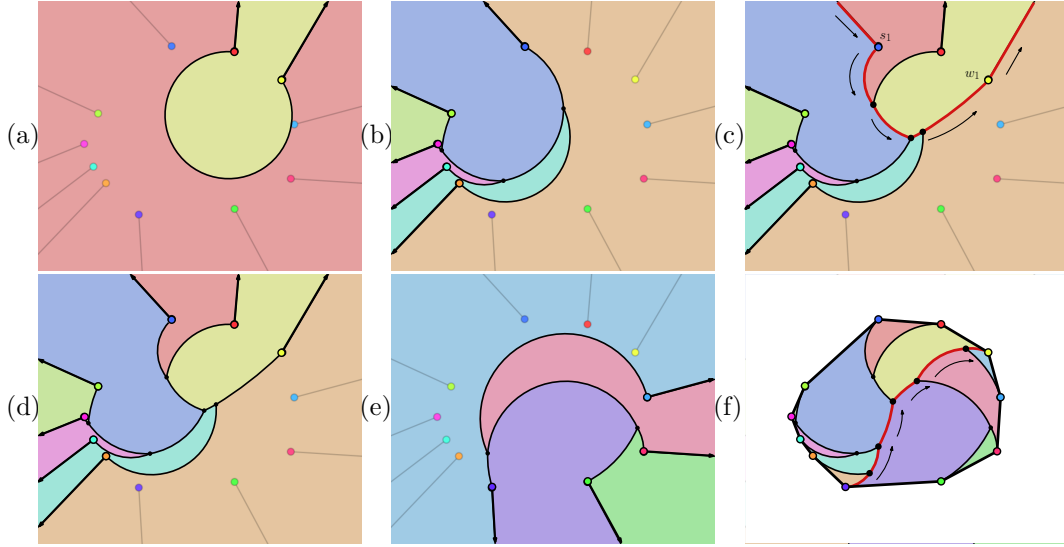


Figure 8 Merging diagrams (a) $\text{RVD}(\mathcal{S}_W^r)$ and (b) $\text{RVD}(\mathcal{S}_S^r)$ into (c) $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$. Merging diagrams (d) $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$ and (e) $\text{RVD}(\mathcal{S}_E^r \cup \mathcal{S}_N^r)$ into (f) $\text{PRVD}(\mathcal{S}_P)$. The highlighted red edges correspond to the merge curve. The arrows schematize tracing.

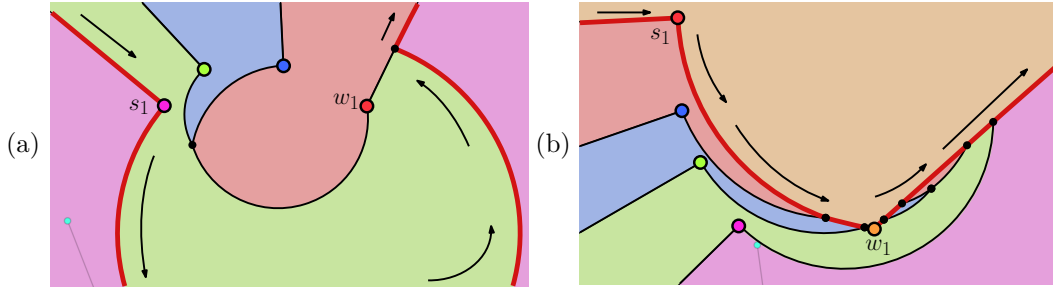


Figure 9 Two special cases of merging two diagrams $\text{RVD}(\mathcal{S}_W^r)$ and $\text{RVD}(\mathcal{S}_S^r)$. (a) The sequence of circular edges does not end at $p(w_1)$. (b) Ray w_1 intersects $\text{RVD}(\mathcal{S}_S^r)$.

- **Tracing w_1 :** In contrast to s_1 , the ray w_1 may intersect many circular edges of $\text{RVD}(\mathcal{S}_S^r)$, each inducing a vertex; see e.g., Figure 9(b). To identify such vertices, we can intersect w_1 (starting from the point on w_1 where E_C ended) with $\text{RVD}(\mathcal{S}_S^r)$. This can be easily done in $O(\mathcal{S}_S^r)$ time, as $\text{RVD}(\mathcal{S}_S^r)$ is a tree.

All steps described can be done in $O(n)$ time, concluding the initial merging step.

Final step. We now merge $\text{RVD}(\mathcal{S}_W^r \cup \mathcal{S}_S^r)$ and $\text{RVD}(\mathcal{S}_E^r \cup \mathcal{S}_N^r)$ restricted to P . The merge curve, as it lies inside P , consists only of E_C , the set of circular edges equidistant to sites $r \in \mathcal{S}_S^r \cup \mathcal{S}_W^r$ and $t \in \mathcal{S}_E^r \cup \mathcal{S}_N^r$. Using the same properties as in the initial step, it follows that E_C consists of a single chain, it does not create bounded faces in $\text{PRVD}(\mathcal{S}_P)$, and tracing may not go outside P . Overall, E_C can be identified in $O(n)$ time, and a simple truncation of the merged diagram inside P , yields $\text{PRVD}(\mathcal{S}_P)$ in $\Theta(n)$ time, concluding the proof.

We can summarize the main result of this section as follows.

► **Theorem 15.** *Given a convex polygon P , we can construct $\text{PRVD}(\mathcal{S}_P)$ in $\Theta(n)$ time.*

4.4 Minimum angle needed to illuminate P - Brocard angle

We now turn back to the problem of computing the Brocard angle α^* of a convex polygon P . As in the respective problem in \mathbb{R}^2 , the angle α^* is realized at a point x^* on $\text{PRVD}(\mathcal{S}_P)$. Since the distance along the edges of $\text{PRVD}(\mathcal{S}_P)$ is monotone, x^* cannot lie on an edge. Further, since $\text{PRVD}(\mathcal{S}_P)$ is confined to P by definition, x^* lies actually on a vertex of $\text{PRVD}(\mathcal{S}_P)$. A special case occurs when α^* is realized by two anti-parallel rays (edges), and the circular arc of their bisector degenerates to a line segment ℓ , as in Figure 3(h). In this case, any point on ℓ is at distance α^* from the rays, but since the endpoints of ℓ are also at distance α^* , the Brocard angle is still realized at a vertex of $\text{PRVD}(\mathcal{S}_P)$.

Therefore, to find α^* we can construct $\text{PRVD}(\mathcal{S}_P)$ and then traverse it to find the vertex of maximum distance. Both steps can be done in $\Theta(n)$ time, so we obtain the following.

► **Theorem 16.** *Given a convex polygon the Brocard angle α^* can be found in $\Theta(n)$ time.*

Following, we give tight bounds on the Brocard angle.

► **Proposition 17.** *Given a convex polygon P , the Brocard angle α^* is less or equal to $\pi/2 - \pi/n$. Further, α^* can take any value in the interval $(0, \pi/2 - \pi/n]$.*

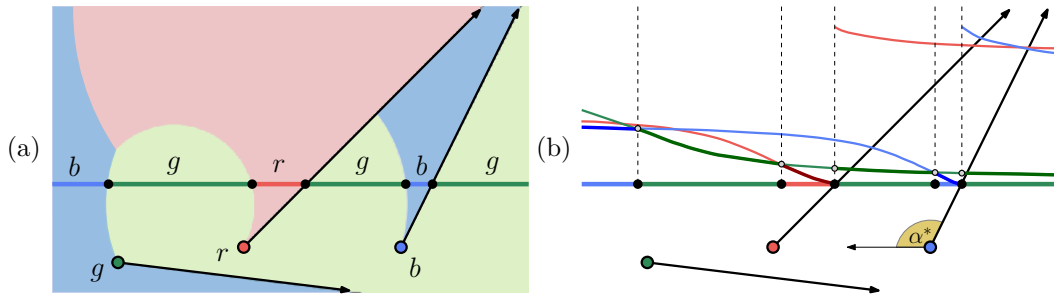
Proof. A $\pi/2 - \pi/n$ upper bound on the Brocard angle is given in [5, 12]. Such an angle is realized by regular polygons. The last illuminated point is the center of the regular polygon, which is simultaneously illuminated by all the floodlights at an angle of $\pi/2 - \pi/n$.

To prove the lower bound, consider a polygon with a bounding box of width w and height h . The aspect ratio $\frac{h}{w}$ can be arbitrarily close to zero, hence α^* is also arbitrarily close to zero. Further, we can smoothly transform any such polygon to a regular polygon, while preserving convexity. Thus, it is possible to get any Brocard angle in the range $(0, \pi/2 - \pi/n]$. ◀

As a final note, observe that the three floodlights, which realize α^* , suffice to illuminate P ; see the dashed segments in Figure 2(a). Consider the k -floodlight illumination problem [28], which asks for the minimum angle α_k , under which a convex polygon is illuminated by k floodlights, $k \geq 3$, where the sum of their apertures is α_k . The Brocard illumination gives an $\alpha_3 = 3\alpha^*$ solution to the problem. By Proposition 17, it follows that $\alpha_3 \leq 3\pi/2 - 3\pi/n$. Hence, for $n \leq 6$, the Brocard angle improves the current $a_3 = \pi$ solution given in [28].

5 RVD on curves

Floodlight illumination problems have also been considered restricted to curves. In this section, we are given a set \mathcal{S} of n rays in \mathbb{R}^2 , a set of α -floodlights each aligned with a ray, and a curve \mathcal{C} , and we want to find the minimum value of α such that \mathcal{C} is completely illuminated



■ **Figure 10** A set \mathcal{S} of 3 rays and the line $\mathcal{C} : x_2 = 0$. (a) $\text{RVD}(\mathcal{S})$ in \mathbb{R}^2 and its intersection with \mathcal{C} . (b) $\text{RVD}(\mathcal{S})$ along \mathcal{C} as the lower envelope (highlighted) of distance functions. The angle α^* is illustrated; the last illuminated point is $(-\infty, 0)$ (first illuminated by the blue ray).

by the set of α -floodlights. We show how these problems can be solved by viewing $\text{RVD}(\mathcal{S})$ as the lower envelope of distance functions in 2-space; see Figure 10. We give the proof for the case when \mathcal{C} is a line and then discuss how this approach can be extended to other curves.

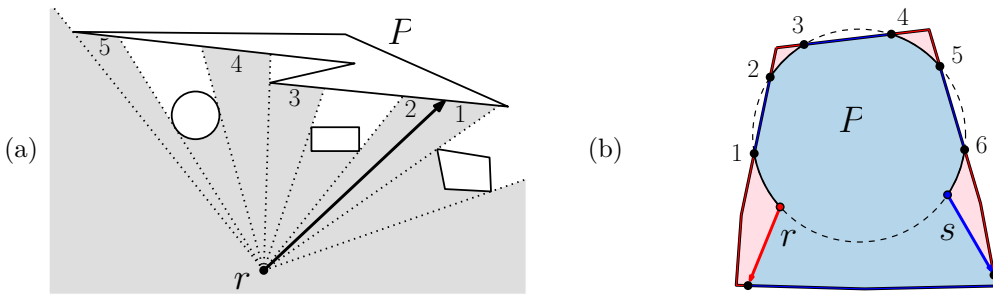
► **Theorem 18.** *Given a line \mathcal{C} , $\text{RVD}(\mathcal{S})$ along \mathcal{C} has complexity $O(n2^{\alpha(n)})$ and it can be constructed in $O(n\alpha(n) \log n)$ time.*

Proof. Without loss of generality, let \mathcal{C} be the horizontal line $x_2 = 0$. Each site $r \in \mathcal{S}$ induces a distance function which maps a point $x = (x_1, 0) \in \mathcal{C}$ to $d_{\angle}(x, r)$; see Figure 10(b). If r intersects \mathcal{C} at point $(i, 0)$, then there is a point of discontinuity, and the distance function is split into two partially defined functions, one with the domain up to i and one with the domain starting at i . $\text{RVD}(\mathcal{S})$ along \mathcal{C} is the lower envelope of these distance functions. The lower envelope of n partially defined functions, where each pair of functions intersects at most s times, has $O(\lambda_{s+2}(n))$ complexity [14] and it can be constructed in $O(\lambda_{s+1}(n) \log n)$ time [15], where $\lambda_s(n)$ is the length of the longest (n, s) Davenport-Schinzel sequence.

In our case, a pair of functions intersects at most twice, as \mathcal{C} may intersect twice the bisecting circle of the two respective rays, so $s = 2$. Further, we have at most $2n$ partially defined functions. Thus, $\text{RVD}(\mathcal{S})$ along \mathcal{C} has complexity $O(n2^{\alpha(n)})$ and it can be constructed in $O(n\alpha(n) \log n)$ time, where $\alpha(n)$ is the inverse Ackermann function. ◀

Observe that, the minimum angle α^* needed to illuminate the line \mathcal{C} is realized either at an intersection point between $\text{RVD}(\mathcal{S})$ and \mathcal{C} , or at a point of \mathcal{C} at infinity; see for example the point $(-\infty, 0)$ in Figure 10. Hence, after constructing $\text{RVD}(\mathcal{S})$ along \mathcal{C} , the angle α^* is revealed by a simple traversal of $\text{RVD}(\mathcal{S})$ in time linear in its size.

The aforementioned approach can be generalized to arbitrary curves in a straightforward way. Let \mathcal{C} be a closed curve enclosing the apices of all the rays of \mathcal{S} . Note that, if \mathcal{C} is a circle, then the same results as Theorem 18 can be derived, since \mathcal{C} intersects a bisecting circle at most twice; hence, $s = 2$. On the other hand, if \mathcal{C} is an m -sided convex polygon, then \mathcal{C} intersects a bisecting circle at most $2m$ times, hence $s = 2m$; see Figure 11(b). If \mathcal{C} is an arbitrary simple polygon, then there might be portions of \mathcal{C} not visible by a ray; thus, an additional parameter k should be introduced, to represent the number of partially defined functions corresponding to each ray.



■ **Figure 11** Illustration of the parameters s and k related to curve illumination. The boundary of the polygon P is the curve to be illuminated. (a) The distance function of the ray r is split into 5 partial functions, i.e., $k = 5$. Splits are induced by visibility constraints due to P itself (breakpoint 3-4), by visibility constraints due to other curves (breakpoint 2-3), or by the discontinuity of the distance function at the ray (breakpoint 1-2). (b) The circular arc of the bisector of the two rays r and s intersects the boundary of P at 6 points, i.e., $s = 6$.

Finally, the approach generalizes to cases where the apices of the rays are not enclosed by some curve; see Figure 11(a). As before, the complexity of $\text{RVD}(\mathcal{S})$ restricted to a curve, will be affected by a parameter s , representing the number of times a curve intersects a bisecting circle, and a parameter k , representing the number of partially defined functions in which a distance function has to be split due to visibility constraints.

6 Concluding remarks

Using the Rotating Rays Voronoi Diagram, we showed how to find the Brocard angle of a convex polygon in optimal linear time, settling an interesting open problem. We exhibited that our method is more general: given any domain D and a set of rays \mathcal{S} , we can find the minimum angle that is required to illuminate D using floodlights aligned with \mathcal{S} , by constructing $\text{RVD}(\mathcal{S})$ restricted to D .

There are many interesting questions to investigate. Regarding the Brocard angle of polygons, we would like to see how our approach could extend to other classes of polygons. We expect to have some difficulties due to the visibility constraints, but we believe that our algorithm could be adapted to work as well. Regarding $\text{RVD}(\mathcal{S})$ in \mathbb{R}^2 , we would like to settle whether the worst case combinatorial complexity is $\Theta(n^2)$. Further, if the current $O(n^{2+\epsilon})$ bound is not tight, we believe that it is possible to design $o(n^{2+\epsilon})$ -time algorithms.

References

- 1 Carlos Alegría-Galicia, David Orden, Carlos Seara, and Jorge Urrutia. Illuminating polygons by edge-aligned floodlights of uniform angle (Brocard illumination). In *Proc. European Workshop on Computational Geometry*, pages 281–284, 2017.
- 2 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013. URL: <http://www.worldscientific.com/worldscibooks/10.1142/8685>.
- 3 Piotr Berman, Jieun Jeong, Shiva P. Kasiviswanathan, and Bhuvan Ugaonkar. Packing to angles and sectors. In *Proc. ACM symposium on Parallel algorithms and architectures*, pages 171–180, 2007.
- 4 Arthur Bernhart. Polygons of pursuit. *Scripta Mathematica*, 24(1):23–50, 1959.
- 5 Ádám Besenyei. The Brocard angle and a geometrical gem from Dmitriev and Dynkin. *The American Mathematical Monthly*, 122(5):495–499, 2015.
- 6 Prosenjit Bose, Leonidas J. Guibas, Anna Lubiw, Mark Overmars, Diane Souvaine, and Jorge Urrutia. The floodlight problem. *International Journal of Computational Geometry & Applications*, 7(1-2):153–163, 1997.
- 7 John Casey. *A sequel to the first six books of the elements of Euclid*. Dublin University Press, 1888.
- 8 Felipe Contreras, Jurek Czyzowicz, Nicolas Fraiji, and Jorge Urrutia. Illuminating triangles and quadrilaterals with vertex floodlights. In *Proc. Canadian Conference on Computational Geometry*, 1998.
- 9 Jurek Czyzowicz, Stefan Dobrev, Benson Joeris, Evangelos Kranakis, Danny Krizanc, Jan Mañuch, Oscar Morales-Ponce, Jaroslav Opatrny, Ladislav Stacho, and Jorge Urrutia. Monitoring the plane with rotating radars. *Graphs and Combinatorics*, 31(2):393–405, 2015.
- 10 Jurek Czyzowicz, Eduardo Rivera-Campo, and Jorge Urrutia. Optimal floodlight illumination of stages. In *Proc. Canadian Conference on Computational Geometry*, pages 393–398, 1993.
- 11 Mark de Berg, Joachim Gudmundsson, Herman Haverkort, and Michael Horton. Voronoi diagrams with rotational distance costs. In *Computational Geometry Week: Young Researchers Forum*, 2017.

- 12 Nikolai A. Dmitriev and E Dynkin. On characteristic roots of stochastic matrices. *Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya*, 10(2):167–184, 1946.
- 13 Vladimir Estivill-Castro, Joseph O’Rourke, Jorge Urrutia, and Dianna Xu. Illumination of polygons with vertex lights. *Information Processing Letters*, 56(1):9–13, 1995.
- 14 Sergiu Hart and Micha Sharir. Nonlinearity of Davenport—Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6(2):151–177, 1986.
- 15 John Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, 1989.
- 16 Dan Ismailescu. Illuminating a convex polygon with vertex lights. *Periodica Mathematica Hungarica*, 57(2):177–184, 2008.
- 17 Rolf Klein. *Concrete and Abstract Voronoi diagrams*, volume 400. Springer Science & Business Media, 1989.
- 18 Rolf Klein, Elmar Langetepe, and Zahra Nilforoushan. Abstract Voronoi diagrams revisited. *Computational Geometry: Theory and Applications*, 42(9):885–902, 2009.
- 19 Rolf Klein and Andrzej Lingas. Hamiltonian abstract Voronoi diagrams in linear time. In *Proc. International Symposium on Algorithms and Computation*, pages 11–19, 1994.
- 20 Evangelos Kranakis, Danny Krizanc, and Oscar Morales. Maintaining connectivity in sensor networks using directional antennae. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, pages 59–84. Springer, 2011.
- 21 Azin Neishaboori, Ahmed Saeed, Khaled A. Harras, and Amr Mohamed. On target coverage in mobile visual sensor networks. In *Proc. ACM International Symposium on Mobility Management and Wireless Access*, pages 39–46, 2014.
- 22 Joseph O’Rourke. Visibility. In *Handbook of discrete and computational geometry*, pages 875–896. CRC Press, 2017.
- 23 Joseph O’Rourke, Thomas Shermer, and Ileana Streinu. Illuminating convex polygons with vertex floodlights. In *Proc. Canadian Conference on Computational Geometry*, pages 151–156, 1995.
- 24 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry*, 12(3):327–345, 1994.
- 25 William Steiger and Ileana Streinu. Illumination by floodlights. *Computational Geometry*, 10(1):57–70, 1998.
- 26 Tsuyoshi Taki, Jun-ichi Hasegawa, and Teruo Fukumura. Development of motion analysis system for quantitative evaluation of teamwork in soccer games. In *Proc. IEEE International Conference on Image Processing*, volume 3, pages 815–818. IEEE, 1996.
- 27 Csaba D. Tóth. Art galleries with guards of uniform range of vision. *Computational Geometry*, 21(3):185–192, 2002.
- 28 Jorge Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027. Elsevier, 2000.

Parallel Computation of Combinatorial Symmetries

Markus Anders

TU Darmstadt, Germany

Pascal Schweitzer

TU Darmstadt, Germany

Abstract

In practice symmetries of combinatorial structures are computed by transforming the structure into an annotated graph whose automorphisms correspond exactly to the desired symmetries. An automorphism solver is then employed to compute the automorphism group of the constructed graph. Such solvers have been developed for over 50 years, and highly efficient sequential, single core tools are available. However no competitive parallel tools are available for the task.

We introduce a new parallel randomized algorithm that is based on a modification of the individualization-refinement paradigm used by sequential solvers. The use of randomization crucially enables parallelization.

We report extensive benchmark results that show that our solver is competitive to state-of-the-art solvers on a single thread, while scaling remarkably well with the use of more threads. This results in order-of-magnitude improvements on many graph classes over state-of-the-art solvers. In fact, our tool is the first parallel graph automorphism tool that outperforms current sequential tools.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Shared memory algorithms

Keywords and phrases graph isomorphism, automorphism groups, algorithm engineering, parallel algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.6

Related Version *Full Version:* <https://arxiv.org/abs/2108.04590>

Supplementary Material *Software:* <https://www.mathematik.tu-darmstadt.de/dejavu> [1]

Funding Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

Acknowledgements We thank Adolfo Piperno, Brendan McKay, Tommi Junttila, and Petteri Kaski for discussions providing us with deeper insights into their isomorphism solvers. We also want to thank our colleagues Thomas Schneider, Jendrik Brächter, and Moritz Lichter for the fruitful discussions we had on some of the topics in this paper.

1 Introduction

Exploitation of symmetry has a dramatic impact on the efficiency of algorithms in various fields. This includes the fields of computer vision and computer graphics [15], automated reasoning [10], machine learning [22] and in particular convolutional neural networks [7], mathematical programming [14], chemical databases [20], SAT-solving [12], constraint programming [8], software verification [5], model checking [9, 18] and so on.

Before symmetries of a structure can be exploited, one first has to have algorithmic means to find the symmetries. For this, the structure is usually transformed into an annotated graph whose automorphisms correspond to the symmetries of the original structure. Then tools are employed that compute the graph's automorphism group.

The current state-of-the-art implementations of solvers computing automorphism groups are BLISS [11], NAUTY and TRACES [17], CONAUTO [16] as well as SAUCY [6]. All of the mentioned algorithms follow the individualization-refinement (IR) framework.



© Markus Anders and Pascal Schweitzer;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 6; pp. 6:1–6:18

Leibniz International Proceedings in Informatics



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

These tools have become increasingly powerful through a multitude of techniques. Initially, each mentioned tool provided insightful new pruning ideas or new implementation tricks. However, many of these very diverse ideas from all the tools have transcended into all of the other tools by now. This lead to a second generation comprised of improved versions of the tools. The most recent and currently fastest solver is TRACES which excels at pruning the search space of difficult graphs. The tool has been meticulously engineered by Piperno over the past decade ever improving its performance. The tool is fastest on most graph classes, and on the few where it is not, it still performs competitively with the best of all the solvers.

Recently the requirements of the application domains for the tools have changed. One major change stems from the different architecture of modern hardware. In fact, all of the aforementioned tools are sequential, single-threaded applications: spreading the work load across multiple cores would align with the contemporary hardware trend of steadily increasing core counts. While there are some theoretical results, research on practical parallel isomorphism algorithms is quite limited. In his thesis, Tener [23] describes approaches to parallel isomorphism testing. However, he has subsequently not pursued this further in the last decade, and the described algorithm is based on algorithms of the first generation. The study [4] only performs comparisons against slow sequential algorithms. Overall, parallel graph isomorphism testing has not witnessed any of the ideas that characterize the second generation of algorithms. Generally, fast isomorphism and automorphism algorithms have been persistently resistant to parallelization attempts in theory and practice.

The goal of this paper is to stimulate a third generation of isomorphism solvers by harnessing the power of a modern CPU for the computation of graph automorphisms. Parallelizing existing libraries is not a straightforward task, since the practical algorithms are based on the IR technique, which is a priori sequential.

Contribution. We introduce DEJAVU, a novel randomized algorithm solving the automorphism group problem based on the IR paradigm. The tool is (1) on a single thread competitive with currently fastest solvers available (sometimes even outperforming them), and (2) on 8 threads outperforms the currently fastest solvers on most graph classes. Using the de facto standard benchmark suite, we report extensive experimental results corroborating these two claims. The results also demonstrate the scalability of the tool as the number of threads is varied from 1 to 8.

Underlying ideas and techniques. The quintessence of our new algorithm, by which we achieve parallelizability, is to replace inherently sequential traversal strategies with randomized traversal: DEJAVU mainly performs repeated random root-to-leaf walks in the search tree (stemming from the IR-framework) in conjunction with a probabilistic abort criterion. The main motivation is that computing multiple random root-to-leaf walks can be parallelized. To allow for an even split of the work load, various subroutines, most notably the so-called sifting algorithm which is used for the probabilistic abort criterion, also have to be parallelized efficiently. However, to create a truly efficient tool, the algorithm has to be combined with further heuristics tailored to the new parallel, probabilistic setting.

Randomization. Our tool DEJAVU is a randomized tool which, in principle, means that the output is not always correct. The idea of exploiting randomization originated from isomorphism testing algorithms [3, 13] which however neither compute automorphisms nor have any form of parallelization whatsoever. For these randomized approaches (including DEJAVU), the user can set an error probability (e.g. 1%) and the tool guarantees that for each

input the probability of error is at most this number. However, crucially our approach only has a 1-sided error. This means while some automorphisms may be missed when queried for the automorphism group of a graph, the solver guarantees that the output consists entirely of automorphisms of the graph. For applications exploiting symmetry this is the *right kind of error*. This way, they may sometimes fail to exploit symmetries that were missed but this only slows down the running time. It does not lead to incorrect results for the application.

2 Preliminaries

2.1 Individualization-Refinement

Following [17] closely, we introduce IR algorithms. The summary is focused on the results necessary to describe automorphism computations and what is needed for our algorithms.

Colored Graphs. An undirected, finite graph $G = (V, E)$ consists of a set of vertices $V \subseteq \mathbb{N}$ and a set of edges $E \subseteq V^2$, where E is symmetric. We always assume $V = \{1, \dots, n\}$.

A coloring $\pi: V \rightarrow \{1, \dots, k\}$ is a surjective map, mapping vertices of a graph to *cells* $1, \dots, k$. We call $\pi^{-1}(i) \subseteq V$ with $i \in \{1, \dots, k\}$ the *i-th cell* of π , which is non-empty since π is surjective. With $|\pi| = k$ we denote the number of cells in a coloring. If $|\pi| = n$ holds, we call π *discrete*. Note that a discrete coloring also characterizes a permutation of V .

A colored graph (G, π) consists of a graph and a coloring. The symmetric group on $\{1, \dots, n\}$ is denoted $\text{Sym}(n)$. With $\text{Aut}(G)$ we denote the automorphism group of a graph. An element $\varphi \in \text{Aut}(G)$ is a permutation of vertices which maps the graph to itself, i.e., a bijective map $\varphi: V \rightarrow V$ where $G^\varphi := (\varphi(V), \varphi(E)) = (V, E) = G$ holds. For colored graphs we additionally require that the coloring is preserved, i.e., a vertex of a cell c must be mapped to a vertex of cell c . We thus define the colored automorphism group $\text{Aut}(G, \pi)$ as those permutations φ with $(G, \pi)^\varphi = (G^\varphi, \pi^\varphi) = (G, \pi)$. Note that in all of these definitions actual equality, e.g., equality of adjacency matrices and not isomorphism, is required. In the following, we only consider uncolored input graphs for the sake of simplicity. Let us remark, however, that we could use exactly the same machinery for colored graphs (see [17]).

Refinement. In the following, we want to *individualize* vertices and *refine* colorings. Individualizing vertices in a coloring is a process that artificially forces the vertex into its own singleton cell. We use $\nu \in V^*$ to denote a sequence of vertices. In particular, we can record in such a sequence which vertices have been individualized.

A *refinement* is a function $\text{Ref}: G \times V^* \rightarrow \Pi$. Here Π is the set of colorings of V , i.e., the set of ordered partitions of V . Given a graph G and sequence of vertices ν , it must satisfy the following properties: first, it is invariant under isomorphism, i.e., $\text{Ref}(G^\varphi, \nu^\varphi) = \text{Ref}(G, \nu)^\varphi$ holds for all $\varphi \in \text{Sym}(n)$. Secondly, it respects vertices in ν as being individualized, i.e., $\{v\}$ is a singleton cell in $\text{Ref}(G, \nu)$ for all $v \in \nu$.

Cell Selector. If refinement classifies all vertices into different cells, determining automorphisms and isomorphisms for a graph is easy, after all, cells have to be preserved. Otherwise, individualization is used to artificially single out a vertex inside a non-singleton class. The task of a *cell selector* is to isomorphism invariantly pick a non-singleton cell of the coloring. In the IR paradigm, all vertices of the selected cell will then be individualized one after the other using some form of backtracking. After the individualization, refinement is applied again and the process continues recursively. Formally, a cell selector is a function $\text{Sel}: G \times \Pi \rightarrow 2^V$ into the power set of V satisfying the following properties:

- It is invariant under isomorphism, that is $\text{Sel}(G^\varphi, \pi^\varphi) = \text{Sel}(G, \pi)^\varphi$ holds for $\varphi \in \text{Sym}(n)$.
- If π is discrete then $\text{Sel}(G, \pi) = \emptyset$.
- If π is not discrete then $|\text{Sel}(G, \pi)| > 1$ and $\text{Sel}(G, \pi)$ is a cell of π .

Search Tree. With the functions Ref and Sel at hand, we are now ready to define the *search tree*. For a graph G we use $T_{(\text{Ref}, \text{Sel})}(G)$ to denote the search tree of G with respect to refinement operator Ref and cell selector Sel. The search tree is constructed as follows: each node of the search tree corresponds to a sequence of vertices of G .

- The root of $T_{(\text{Ref}, \text{Sel})}(G)$ is the empty sequence ϵ .
- If ν is a node in $T_{(\text{Ref}, \text{Sel})}(G)$ and $C = \text{Sel}(G, \text{Ref}(G, \nu))$, then its children are $\{\nu.v \mid v \in C\}$, i.e., all extensions of ν by one vertex v of C .

With $T_{(\text{Ref}, \text{Sel})}(G, \nu)$ we denote the subtree of $T_{(\text{Ref}, \text{Sel})}(G)$ rooted in ν . We omit indices Sel and Ref if they are apparent from context. Note that the leaves of a search tree correspond to discrete colorings of the graph, and therefore to permutations of V .

We recite the following crucial facts on isomorphism invariance of the search tree as given in [17], which follows from the isomorphism invariance of Sel and Ref:

► **Lemma 1.** *For a graph G and $\varphi \in \text{Sym}(n)$ we have $T(G)^\varphi = T(G^\varphi)$.*

► **Corollary 2.** *If ν is a node of $T(G)$ and $\varphi \in \text{Aut}(G)$, then ν^φ is a node of $T(G)$ and $T(G, \nu)^\varphi = T(G, \nu^\varphi)$.*

We have yet to mention how the search tree is used to find automorphisms of a graph:

► **Lemma 3.** *If ν and ν' are leaves of $T(G)$, then there exists an automorphism $\varphi \in \text{Aut}(G)$ such that $\nu = \varphi(\nu')$ if and only if $\text{Ref}(G, \nu')^{-1} \cdot \text{Ref}(G, \nu)$ is an automorphism of G .*

We also say that ν' is an *occurrence* of ν if there is some automorphism $\varphi \in \text{Aut}(G)$ for which $\varphi(\nu') = \nu$.

Pruning. In the overall algorithm, we fix a single leaf τ and then search for automorphisms by comparing other leaves to it. We call this fixed leaf τ the *target leaf*. Corollary 2 and Lemma 3 show that this suffices to derive all automorphisms from the search tree.

Unfortunately, however, the search tree itself can be exponentially large in the input [19]. Therefore, we want to prune it as much as possible.

Towards this goal, we define a *node invariant* $\text{Inv}: G \times V^* \rightarrow I$, which is a function mapping nodes of the tree to a totally ordered set I . We require some further properties:

- The invariant must be isomorphism invariant, i.e., we require $\text{Inv}(G, \nu_1) = \text{Inv}(G^\varphi, \nu_1^\varphi)$ for all $\varphi \in \text{Sym}(n)$.
- If $|\nu_1| = |\nu_2|$ and $\text{Inv}(G, \nu_1) < \text{Inv}(G, \nu_2)$, then for all leaves $\nu'_1 \in T(G, \nu_1)$ and $\nu'_2 \in T(G, \nu_2)$ we require $\text{Inv}(G, \nu'_1) < \text{Inv}(G, \nu'_2)$.

It follows that even if we remove all nodes of the tree whose invariant deviates from the corresponding node invariant on the same level on the path to the target leaf, we can still retrieve the entire automorphism group. This operation is called *pruning using invariants*. Formally, we define $\text{Prune}_{\text{Inv}}(\tau', \nu')$ to denote the operation that removes the subtree of node ν' if $\text{Inv}(G, \tau') \neq \text{Inv}(G, \nu')$, where $|\tau'| = |\nu'|$ holds and τ' is the prefix of length $|\nu'|$ of τ .

We now describe *pruning using automorphisms*. Assume we already have $\varphi \neq \text{id}$ of $\text{Aut}(G)$ available. For nodes ν where ν^φ is not a prefix of the target leaf, we define $\text{Prune}_{\text{Aut}}(\nu, \nu^\varphi)$ to denote the operation which removes the subtree rooted at ν^φ from the search tree. Applying $\text{Prune}_{\text{Aut}}$ can only cut away parts of the search tree which are generated by the already available automorphisms anyway [17].

■ **Algorithm 1** Sifting.

```

1 function Sift( $S, T, B, \varphi$ )
    Input : generators  $S$ , transversal table  $T$ , base points  $B$ , element  $\varphi$ 
    Output : whether  $S, B$  and  $T$  remained unchanged
2 for ( $i = 1; i \leq |B|; i = i + 1$ )
3      $b_i := \varphi(B_i);$ 
4      $t := (T_i)_{b_i};$ 
5     if  $t = \perp$  then break;
6      $\varphi := \varphi \cdot t^{-1};$ 
7 if  $\varphi \neq \text{id}$  then
8      $S := S \cup \{\varphi\};$ 
9      $b_i := \varphi(B_i);$ 
10     $(T_i)_{b_i} := \varphi;$ 
11    return false;
12 return true;

```

2.2 Schreier-Sims Fundamentals

The procedure to aggregate automorphisms of DEJAVU works on similar principles as the random Schreier-Sims algorithm, which provides us with a data structure to dynamically manage permutation groups. To be more precise, our algorithm needs a way to determine whether a newly found automorphism φ is in the group generated by the automorphisms that were found previously. The procedure we use for this is called *sifting*. We give a brief description following the lines of [21].

All groups we consider are permutation groups $\Gamma \leq \text{Sym}(\Omega)$. For the domain we always set $\Omega = \{1, \dots, n\}$. By $\langle S \rangle$ we denote the group generated by the elements of S , i.e., all elements that can be written as a product of elements of S . If $\langle S \rangle = \Gamma$ holds, we call S a *generating set* of Γ .

We need the notion of a *pointwise stabilizer* of a permutation group $\Gamma \subseteq \text{Sym}(\Omega)$. Let $\beta \in \Omega$ be a point, then $\Gamma_{(\beta)} := \{\varphi \in \Gamma \mid \varphi(\beta) = \beta\}$. For a sequence of points $(\beta_1, \dots, \beta_m) \in \Omega^m$ we just recursively take the pointwise stabilizer of all elements:

$$\Gamma_{(\beta_1, \dots, \beta_m)} := \begin{cases} \Gamma & \text{if } m = 0 \\ (\Gamma_{(\beta_1, \dots, \beta_{m-1})})_{(\beta_m)} & \text{otherwise.} \end{cases}$$

We call a sequence of points $B = (\beta_1, \dots, \beta_m) \in \Omega^m$ a *base* relative to $\Gamma \leq \text{Sym}(\Omega)$ if $\Gamma_B = \{\text{id}\}$. For a generating set $\langle S \rangle = \Gamma$ and a base $(\beta_1, \dots, \beta_m)$ we define $S_i = S \cap \Gamma_{(\beta_1, \dots, \beta_i)}$. We call S *strong* relative to Γ and $(\beta_1, \dots, \beta_m)$ if $\langle S_i \rangle = \Gamma_{(\beta_1, \dots, \beta_i)}$ holds for all $i \in \{0, \dots, m\}$.

Given a subgroup $\Delta \leq \Gamma$, a *transversal* of Δ in Γ is a subset $T \subseteq \Gamma$ that satisfies $|T \cap gH| = 1$ for every coset $g\Delta$ of Δ in Γ . We construct a *transversal table* for a given base B and generating set S , which contains a transversal for each subgroup $\langle S_i \rangle$ in $\langle S_{i-1} \rangle$. We refer with T_i to the transversal of S_i . Careful inspection of the definition reveals that each $\langle S_i \rangle$ fixes the i -th base point of B , i.e., for all $\varphi \in \langle S_i \rangle$ it is true that $\varphi(\beta_i) = \beta_i$. If we want to know the cosets of S_i in $\langle S_{i-1} \rangle$, we need to find the possible images of β_i in $\langle S_{i-1} \rangle$. Elements of $\langle S_{i-1} \rangle$ under which β_i has the same image are in the same coset of $\langle S_i \rangle$. Thus, we can differentiate transversal elements T_i according to the image of β_i under them. We denote by $(T_i)_b$ the element in T_i mapping $\beta_i \mapsto b$ if it exists. We set $(T_i)_b = \perp$ if such an element does not exist. The cosets correspond to the orbit of β_i in S_{i-1} . Given an element

$\varphi \in S_{i-1}$, we need to determine to where φ maps β_i in order to find the coset in which it is contained. The representative of the coset is that element t in the transversal T_i which also maps β_i to $\varphi(\beta_i)$. By forming the product $\varphi \cdot t^{-1} \in S_i$ we obtain an element that fixes β_i .

Algorithm 1 describes a *sifting* procedure, which can be used to test membership in a given permutation group whenever a strong generating set S and corresponding base B are available. Otherwise, if S is not strong or B not complete, the sifting procedure computes a non-trivial permutation. In the version of the algorithm described here, this permutation is added to the generating set to ensure that now the sifted element is covered. Possibly one needs to extend the base for this purpose. If an element *sifts successfully*, i.e., the procedure returns *true*, we know that it is contained in $\langle S \rangle$. On the other hand, if the sifting is unsuccessful then the element was not in the group or the generating set was not strong and has been extended towards ensuring it to become strong.

The algorithm repeatedly multiplies transversal elements to the initial element. The operations preserve the property of whether the initially given element is in the group. Each operation modifies the element so that it is contained the next respective pointwise stabilizer.

We refer to base, transversal table and generating set together as a *Schreier structure*. As elements are sifted, such a structure captures the progress made towards constructing the group. A crucial result we exploit is the following, related to Lemma 4.3.1 in [21]:

► **Lemma 4.** *Let Γ be a group, B a base, S a set of permutations in Γ and φ a uniformly distributed element in Γ . If $\langle S \rangle \neq \Gamma$, the probability that φ does not successfully sift through the Schreier structure defined by B and S is at least $\frac{1}{2}$.*

The previous results are also the foundation for the random Schreier-Sims method, which is used by all competitive solvers to detect possibilities to apply the pruning function $\text{Prune}_{\text{Aut}}$.

Let us also record that the individualized vertices in a leaf of the search tree (defined in Section 2.1) actually form a base of the respective automorphism group [17].

3 Parallel Computation of Automorphisms

We first describe how to turn random walks on IR trees into a correct, probabilistic algorithm. Then, we discuss how to parallelize sifting as required by the algorithm. Lastly, we augment the algorithm using breadth-first traversal into the underlying procedure of DEJAVU .

The motivation is that the three fundamental methods mentioned above parallelize efficiently as long as the IR tree is sufficiently large.

3.1 Random Walks and Automorphisms

The first step of our algorithm is to compute a random walk to one of the leaves, the target leaf. The goal is then to find another occurrence target leaf through random walks, whereby automorphisms are found. A key observation is that by choosing uniform, random walks through the tree – which we describe in Algorithm 2 – we also get a uniform distribution of elements in the automorphism group. The algorithm applies the refinement to the input graph and then repeatedly chooses a uniform random vertex of the target cell chosen by the cell selector for individualization. Starting from the initial coloring, it then keeps individualizing and refining until the coloring becomes discrete. It returns the coloring and the sequence of individualized vertices.

Recall that we refer to a leaf τ' as an *occurrence* of τ if τ' can be mapped to τ using an element $\varphi \in \text{Aut}(G)$ (i.e., $\varphi(\tau') = \tau$). In this situation we call φ the *corresponding automorphism* with regard to τ' . Note that there is a unique occurrence of τ for every $\varphi \in \text{Aut}(G)$:

► **Lemma 5.** *A leaf τ can be mapped to exactly $|\text{Aut}(G)|$ leaves in $T(G)$ using elements of the automorphism group $\text{Aut}(G)$.*

Proof. Note that τ is a base of $\text{Aut}(G)$. Now consider an element $\varphi \in \text{Aut}(G)$. Clearly, τ^φ also corresponds to a leaf in the tree (Lemma 2) and τ^φ is a base as well. Now consider a different element $\varphi' \in \text{Aut}(G)$, i.e., $\varphi' \neq \varphi$. Clearly, $\tau^\varphi \neq \tau^{\varphi'}$ holds since τ is a base. ◀

► **Lemma 6.** *As a random variable, the output of Algorithm 2, which is a leaf in the search tree, is uniformly distributed within each equivalence class of leaves.*

Proof. There is a unique occurrence of τ for every automorphism (Lemma 5). Hence, it suffices to argue that the probability of finding each occurrence of τ through a random walk in the tree is equal. Assume that we are in a node ν of the search tree and let ν_1, \dots, ν_k be the children of ν . Let ν'_1, \dots, ν'_k be the children that correspond to the subtrees of ν that do contain an occurrence of τ . Since we are sampling an element uniformly from ν_1, \dots, ν_k in Algorithm 2, each of these subtrees has the same probability of being chosen. Therefore, it suffices to argue that the chance of finding an occurrence of τ in each of ν'_1, \dots, ν'_k is equal. Since they all contain an occurrence of τ , they can all be mapped to each other using the corresponding automorphisms. But this immediately implies that all of these subtrees must be isomorphic (Lemma 1), showing the claim. ◀

The following lemma immediately follows.

► **Lemma 7.** *Let τ be a fixed leaf. Consider the distribution of outputs of Algorithm 2 under the condition that an occurrence of τ is computed. For such a given output τ' consider the automorphism φ with $\varphi(l) = l'$ corresponding τ' . Then φ is uniformly distributed in $\text{Aut}(G)$.*

So, Algorithm 2 provides us with a method to uniformly sample random automorphisms. We now need a method to collect these automorphisms and determine when we have found enough of them to generate the entire automorphism group.

Description of Algorithm 3. The algorithm repeatedly samples automorphisms from the automorphism group through random walks (using Algorithm 2). Then, it uses a probabilistic test based on Lemma 4 and Lemma 7 to determine termination. When a certain number $d = \lceil -\log_2(\frac{\varepsilon}{2}) \rceil$ of *consecutively* sampled automorphisms turn out to be already covered by the previously found automorphisms (i.e., they sift successfully) the algorithm terminates. The initial value of d is linked to the guaranteed bound on the error probability ε that can be chosen by the user. To guarantee that the error bound is kept, when some but not d consecutively found automorphisms were discovered, the value of d is incremented.

Finishing the execution therefore hinges on seeing already explored leaves as well as already generated automorphisms again. Note that the correctness of the algorithm depends on the fact that we are probing automorphisms uniformly from the group. In Section 3.3, we introduce further techniques to prune the search tree. When we do so, we always make sure to do this in a manner that still enables us to probe uniformly after the pruning. Ensuring this suffices to retain a correct behavior of the algorithm.

We now argue correctness for Algorithm 3.

► **Lemma 8.** *Given a graph G and probability ε , Algorithm 3 produces a generating set for the automorphism group of G with probability at least $1 - \varepsilon$.*

Proof. First, observe that the discovered permutations are certified before being added to the group, which immediately ensures that all elements of the computed group are actual automorphisms. The algorithm can therefore only fail by not adding enough elements to the group.

■ **Algorithm 2** Random Walk of the Search Tree.

```

1 function RandomWalk( $G$ )
  Input : graph  $G$ 
  Output : a random leaf of the search tree and the individualized vertices
2   $base := ()$ ;
3   $col := \text{Ref}(G, [v \mapsto 1], base)$ ;
4   $cell := \text{Sel}(G, col)$ ;
5  while  $cell \neq \emptyset$  do
6     $v := \text{RandomElement}(cell)$ ;
7     $base := base.v$ ; // append  $v$  to  $base$ 
8     $col := \text{Ref}(G, col, base)$ ;
9     $cell := \text{Sel}(G, col)$ ;
10 return ( $col, base$ );

```

■ **Algorithm 3** Parallel Randomized Automorphisms.

```

1 function Automorphisms( $G, \epsilon$ )
  Input : graph  $G$  and probability  $\epsilon$ 
  Output : a subset of  $\text{Aut}(G)$  that generates  $\text{Aut}(G)$  with probability at least  $1 - \epsilon$ 
2   $c := 0, d := \lceil -\log_2(\frac{\epsilon}{2}) \rceil, S := \emptyset$ ;
3   $(\tau, B) := \text{RandomWalk}(G)$ ;
4  initialize trivial transversal table  $T$  relative to  $B$ ;
5  while  $c \leq d$  do in parallel
6    // run multiple instances of the body of the loop in parallel
7     $(l', \_) := \text{RandomWalk}(G)$ ;
8     $\varphi := l' \cdot \tau^{-1}$ ;
9    if  $\varphi(G) = G$  then
10     if  $\neg \text{Sift}(S, T, B, \varphi)$  then  $c := c + 1$ ;
11     else
12       if  $c > 0$  then  $d := d + 1$ ;
13        $c := 0$ ;
14 return  $S$ ;

```

Choosing random walks through the tree produces a uniform distribution of occurrences of the leaf τ , which gives us a uniform distribution of elements in $\text{Aut}(G)$ (Lemma 7). This in turn enables us to use Lemma 4 to argue correctness as follows.

We terminate the algorithm after we sifted uniform elements of G successfully into the Schreier structure d times in a row. As long as sifting fails and we add elements to the Schreier structure, we know that no error occurs and that we are not done. We view the computation as a sequence of *tests* against the hypothesis that we are missing automorphisms. We define the beginning of a test to be right after sifting succeeds once (i.e., at the moment when c is set to 1 in an execution of Line 10). The probability that the test fails (i.e., that we do not abort the test early and instead increment c for d times in a row) is bounded by $(\frac{1}{2})^d$ (Lemma 4). In order to ensure a total error bound of ϵ for the algorithm, we require the sum of the probabilities of the tests to fail is at most ϵ . For this it suffices to have that the i -th test fails with probability at most $\frac{\epsilon}{2^i}$. The probability that the entire computation fails is then surely at most ϵ since $\sum_{i=1}^{\infty} \frac{\epsilon}{2^i} \leq \epsilon$.

In order to satisfy this bound of $\frac{\varepsilon}{2^i}$, we increment d after each successful test. Initially, for the first test, we set $d_1 = \lceil -\log_2(\frac{\varepsilon}{2}) \rceil$ which ensures that $(\frac{1}{2})^d \leq \frac{\varepsilon}{2}$. Note the value d_i for variable d used during the i -th test is then $d_i = d_1 + i - 1$, so $(\frac{1}{2})^{d_i} < \frac{\varepsilon}{2^i}$, as desired. ◀

We should clarify that while the algorithm is based on some of the same principles as the isomorphism test of [3], that isomorphism test neither has to consider uniformity of automorphism sampling (Lemma 7), nor employ repeated testing, nor requires any form of sifting.

Through the use of the randomized approach, a simple opportunity for parallelization arises by running the body of the while-loop in Line 6 on multiple threads. In particular, only two components have to be synchronized: the state of the abort criterion c and d , as well as the Schreier structure S and T which is manipulated by the sifting procedure. While the former is trivial, parallelization of the sifting procedure is discussed in the following section.

There is a slight technical issue we should address when running Algorithm 3 in parallel. If, say, the elements that are already generated by S can be computed more quickly than those that are not, using many threads would create a bias towards finding the former type of element first, leading to an incrementation of c with a probability larger than $1/2$. This would break the error bound. However, there is a simple way to fix the issue: whenever c exceeds d , it suffices to additionally ensure all threads finish their current iteration.

3.2 Sifting in Parallel

For the abort criterion of the algorithm, we check whether an automorphism is contained in the group generated by the automorphisms found so far (see Line 10 of Algorithm 3). To check this, we sift it into a Schreier structure using a base of the automorphism group.

As it turns out, sifting elements is sometimes expensive: using a conventional, sequential implementation of the sifting procedure to determine the abort criterion of our algorithm does not scale with more threads. In practice, sifting would often become the bottleneck.

For the random abort criterion we observe the following when sifting elements.

1. The base is never changed or extended.
 2. Changes in the transversal tables T are always local to one level in the Schreier structure.
 3. In practice, if sifting is expensive, many elements are sifted. The computationally expensive part is then mostly multiplication of elements (Line 6 of Algorithms 1 and 4).
- We should stress that in particular, (1) and (2) are generally *not true* when sifting is employed by previous IR algorithms, and are indeed specific to the way it is used by Algorithm 3.

Crucially, these three observations enable a rather simple modification to the algorithm: we can sift elements into a Schreier structure concurrently, as long as we synchronize changes to transversal tables when changing a level. We add a lock for every level and one global lock for the generating set to enable parallel sifting on a fixed base (see Algorithm 4). We should remark that the locking mechanism could be made more granular to further improve scaling. However, due to observation (3), we never deemed this necessary in practice.

3.3 Uniform Pruning

While Algorithm 3 is able to solve the problem on its own and parallelizes whenever enough random walks are required, it never actually prunes the search tree. This means that if it is discovered during random probing that a certain path does not lead to an occurrence of the target leaf, there is no mechanism to prevent making the same bad choices again.

■ **Algorithm 4** Thread-safe Sifting.

```

1 function Sift( $S, T, B, \varphi$ )
  Input : generators  $S$ , transversal table  $T$ , base  $B$ , element  $\varphi$ 
  Output : whether  $S$  and  $T$  remained unchanged
2 for (  $i = 1$ ;  $i \leq |B|$ ;  $i = i + 1$  )
3    $b_i := \varphi(B_i)$ ;
4    $t := (T_i)_{b_i}$ ;
5   if  $t = \perp$  then break;
6    $\varphi := \varphi \cdot t^{-1}$ ;
7 if  $\varphi \neq \text{id}$  then
8   acquire lock for level  $i$ ;
9   acquire lock for generators;
10   $S := S \cup \{\varphi\}$ ;
11  release lock for generators;
12   $b_i := \varphi(B_i)$ ;
13  update  $(T_i)_{b_i} = \varphi$ ;
14  release lock for level  $i$ ;
15  return false;
16 return true;

```

To rectify this, we want to intersperse the random walks of the tree with breadth-first search. Fortunately, probing after breadth-first traversal of an entire level has been performed naturally results in a uniform distribution of leaves again. Indeed, upon completion of an entire level of breadth-first traversal, probing can also be characterized as starting a random walk at a uniformly random node of that level. We probe from a level k by choosing uniformly at random a node ν' with $|\nu'| = k$ of the search tree that satisfies $\text{Inv}(\nu') = \text{Inv}(\nu)$. Here ν is the prefix of length k of the vertex sequence corresponding to the target leaf. Due to Lemma 1, the trees rooted in prefixes that contain some occurrence of the target leaf are isomorphic. Therefore they yield the same probability for finding an occurrence of the target leaf within them. The process therefore samples automorphisms with a uniform distribution. Hence, breadth-first traversal can safely be combined with Algorithm 3.

When using the breadth-first traversal, we still want to be able to use the automorphism pruning rule $\text{Prune}_{\text{Aut}}$. However, this can break uniformity. Assume for example ν and ν' are both children of μ' . Assume further they each correspond to a “bad choice” on this level of the search tree in the sense that neither of them contains an occurrence of the target leaf. During the algorithm we would not yet know whether these choices are bad, but suppose we find an automorphism mapping one to the other. By contracting them to a single node, we reduce the number of children without a target leaf and thus increase the chance of finding an occurrence of the target leaf in μ' .

Our solution to this problem is that whenever we use $\text{Prune}_{\text{Aut}}$, we artificially restore uniformity. We do so by introducing *weights* to nodes of the tree, which essentially denote the number of paths represented by a node of the tree. When combining elements of the same level, their weights (represented paths) are combined as well. Later on, when probing for leaves, we take weights into account when sampling random elements. Considering the example again, when we contract the bad choices ν and ν' to a single node, and both have weight 1 to begin with, the remaining node gets weight 2. The remaining node is then chosen with the same probability as both of the initial nodes together – hence, keeping the same probability of finding the target leaf in μ .

Since the probability of finding automorphisms is supposed to remain constant, one might wonder why this kind of pruning should be applied at all. The reason is that the work for the breadth-first traversal is reduced: we need to compute less nodes when advancing the breadth-first level, since symmetric nodes are contracted. On later levels, we may be able to throw away nodes (uniformly, since we are performing breadth-first traversal) and thus actually increase the probability of finding target leaves.

We now formalize the notion of weights. We describe this using the following construction: we start by defining internal weights \bar{w} and external weights w for all nodes. The internal weights and external weights of all nodes are initially set to 1, i.e., $w(\nu) := \bar{w}(\nu) := 1$ for all ν . The internal weights are then manipulated by the algorithm. Whenever internal weights are modified, the following formula calculates the corresponding external weights:

$$w(\nu) := \begin{cases} 1 & \text{if } \nu = \epsilon \\ \bar{w}(\nu) \cdot w(\nu_1, \dots, \nu_{k-1}) & \text{if } \nu = \nu_1, \dots, \nu_k \end{cases}$$

We now modify $\text{Prune}_{\text{Aut}}$ into $\text{Prune}'_{\text{Aut}}$ by making it update the internal weights in addition to pruning: assume we already have φ of $\text{Aut}(G)$ available. For all nodes ν where ν^φ is not a prefix of the target leaf, $\text{Prune}'_{\text{Aut}}(\nu, \nu^\varphi)$ removes ν^φ from the search tree and updates $\bar{w}(\nu) = \bar{w}'(\nu) + \bar{w}'(\nu^\varphi)$, where \bar{w}' are the previous internal weights.

Additionally, we want to formalize the notion that the tree rooted in ν^φ is now *represented* by the tree rooted in ν . We do this by introducing an equivalence relation \sim , which we update every time $\text{Prune}_{\text{Aut}}$ is executed. Initially, every node represents itself, hence $[\nu]_\sim = \{\nu\}$ holds. Note that trivially $|\llbracket \nu \rrbracket_\sim| = w(\nu) = 1$ is satisfied initially. We update the relation whenever $\text{Prune}'_{\text{Aut}}(\nu, \nu^\varphi)$ is executed. The old relation \sim' is then replaced by \sim , which we define in the following. We do so using three states of the search tree: first, the *unpruned tree* $T(G, \nu)$, which is the initial tree where no pruning rule has been applied. Secondly, there is the *pruned tree before* the operation $\text{Prune}'_{\text{Aut}}$ was executed, i.e., $T(G, \nu)'$ with the relation \sim' . Lastly, there is the *pruned tree after* the operation $\text{Prune}'_{\text{Aut}}$ has taken place, for which we want to define the relation \sim . The goal is then to argue inductively that the size of the equivalence class is equal to the external weight of the representative in the pruned trees.

We stipulate that ν^φ , as well as all nodes currently represented by ν^φ , are now represented by ν . Formally, we unify the equivalence classes of the root nodes in question, i.e., we set $[\nu]_\sim := [\nu]_{\sim'} \cup [\nu^\varphi]_{\sim'}$.

We extend this definition recursively for all nodes of the tree rooted in ν , i.e., all elements of the unpruned tree rooted in ν^φ need to be represented by some element of $\nu' \in T(G, \nu)'$. The tree rooted at ν^φ may also represent other trees, which need to be considered. We let $[\nu^\varphi]_{\sim'} = \{\nu^{\varphi_1}, \dots, \nu^{\varphi_k}\}$, where clearly $\varphi_i \in \langle S \rangle$ for $i \in \{1, \dots, k\}$ holds. Since all of these trees may have been pruned differently through applications of $\text{Prune}'_{\text{Aut}}$, we refer back to nodes of the unpruned tree $T(G)$. For every node ν' in the pruned tree $T(G, \nu)'$, we find all of the nodes of the unpruned tree that are now represented by it, i.e., $[\nu']_\sim := [\nu']_{\sim'} \cup \{\nu''^{\varphi_i} \mid \nu'' \in [\nu']_{\sim'}, i \in \{1, \dots, k\} \mid \nu'' \in T(G, \nu)\}$. Note that this is well-defined in terms of equivalence relations, since all φ_i define bijections between nodes of $T(G, \nu)$ and $T(G, \nu^{\varphi_i})$ (Lemma 2). All other nodes of the pruned tree keep their equivalence classes of \sim' , which is correct since their weight is unaltered as well.

We now argue that the external weight of a remaining node represents the size of its respective equivalence class:

► **Lemma 9.** *Let A be a sequence of applications of $\text{Prune}'_{\text{Aut}}$ to a search tree $T(G)$. Assume that applying A results in $T(G)'$ with external weight function w' and \sim' is the corresponding equivalence relation (as described previously). Then, it holds that $|\llbracket \nu \rrbracket_\sim| = w'(\nu)$.*

Proof. We argue the claim by induction over the sequence A . Initially, the claim is true since all weights are 1 and every node represents itself in \sim .

Let us now argue that when applying some $\text{Prune}'_{\text{Aut}}(\nu, \nu^\varphi)$, the invariant remains valid. For the root node this is easy to see: by induction $|\nu]_{\sim}| = w'(\nu)$ and $|\nu^\varphi]_{\sim}| = w'(\nu^\varphi)$ hold, which in turn shows $|\nu]_{\sim}| = |\nu]_{\sim} \cup [\nu^\varphi]_{\sim}| = w'(\nu) + w'(\nu^\varphi) = w(\nu)$. The third equality holds since $\varphi \neq \text{id}$ is required by definition. We now show this for all $\nu' \in T(G, \nu)'$ by induction. Let $\nu' = \nu'_1, \dots, \nu'_k$ and $\mu = \nu'_1, \dots, \nu'_{k-1}$. We know that $w(\mu) = |\mu]_{\sim}|$ holds, i.e., the statement is true for the parent of ν' .

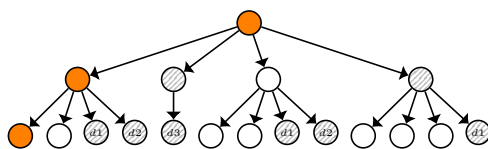
It can be easily seen that set of the elements $\mu' \in T(G, \nu)$ with $\mu' \in [\mu]_{\sim}$, i.e., elements represented by μ in the subtree of ν , has not been altered. Hence, we can rewrite $w(\mu) = |\mu]_{\sim}| = |\{\mu' \in [\mu]_{\sim} \mid \mu' \in T(G, \nu)\}| \cdot w(\nu)$. The internal weight \bar{w} of ν' is only changed whenever $\text{Prune}'_{\text{Aut}}$ is directly applied on ν' . This means that in the unpruned tree, ν' represents $\bar{w}(\nu')$ many elements in $T(G, \mu)$. We can conclude $|\nu']_{\sim}| = |\{\nu'' \in [\nu']_{\sim} \mid \nu'' \in T(G, \nu)\}| \cdot w(\nu) = \bar{w}(\nu') \cdot |\{\mu' \in [\mu]_{\sim} \mid \mu' \in T(G, \nu)\}| \cdot w(\nu) = \bar{w}(\nu') \cdot w(\mu)$, which proves our claim. \blacktriangleleft

Since we weigh each equivalence class of nodes with its size in the pruned tree, it does not matter up to \sim whether we perform random walks on the pruned search tree or the unpruned tree. The distributions of equivalence classes are indistinguishable. Let us now argue why this suffices for the correctness of our algorithm. By looking carefully at the previous discussion, we can observe that all elements of an equivalence class can be mapped to each other through elements of $\langle S \rangle$. This implies that given a leaf ν , all elements represented by ν are generated by S if and only if ν is generated by S . In terms of Lemma 4, the automorphism derived from a leaf sampled uniformly at random from ν therefore has the same chance of sifting through the structure as one of ν^φ . Therefore, by using a modified version of Lemma 4 as the abort criterion, we can consider sampled weighted nodes of the pruned tree instead of proper uniform random nodes of the unpruned tree:

► **Lemma 10.** *Let G be a graph, B a base, S a set of permutations, $T(G)'$ the search tree resulting from repeated application of $\text{Prune}'_{\text{Aut}}$ with elements of S and $\tau \in T(G)'$ the target leaf. Furthermore, let ν be a leaf drawn from $T(G)'$ with weight $w(\nu)$ where $\nu^{-1} \cdot \tau = \varphi \in \text{Aut}(G)$. If $\langle S \rangle \neq \text{Aut}(G)$ holds, then the probability that φ does not sift through the Schreier structure defined by B and S is at least $\frac{1}{2}$.*

Proof. Since ν is an occurrence of the target leaf, we can require $\nu^\varphi = \tau$. From the previous discussion, we know that external weights of remaining leaves determine the amount of leaves in the unpruned tree represented by them (Lemma 9) and that no other remaining leaves represent them. It therefore suffices to argue that the derivable automorphisms of leaves of $T(G)$ represented by ν all have the same chance of being generated by $\langle S \rangle$. Since leaves are represented by ν if they were pruned using $\text{Prune}'_{\text{Aut}}$ – which by assumption can only use elements of S – all of them can be derived by applying some $\varphi' \in \langle S \rangle$ to ν : hence, $\varphi \in \langle S \rangle \iff \varphi \cdot \varphi' \in \langle S \rangle$. \blacktriangleleft

The actual solver will – in addition to finding a generating set in the first place – still need to fill the Schreier structure sufficiently. This can however only require more elements and thus increases the chance of elements not sifting successfully. Hence, it even decreases error probabilities.



■ **Figure 1** Potential search tree traversed when using trace deviation sets. Orange indicates base nodes and gray indicates pruned nodes. Children of pruned nodes are of course eventually pruned as well.

4 Implementation and Heuristics

We use the tools of the previous section to construct the parallel graph automorphism solver DEJAVU. We start by describing the high-level structure of DEJAVU. It consists of 4 modes of operation, between which it continuously switches. The solver decides how to switch between the modes using heuristics, which are based on a cost estimation.

The solver begins by trying to sample a good cell selector (in parallel). Then, *base-aligned* automorphism search is performed (Section 4.2), followed by breadth-first search (Section 4.1), followed by *level* automorphism search (see Section 4.2). At any point, depending on the cost estimation, the solver can decide to go to a preceding mode again.

The tool is written in C++. Threads of the C++ standard library are used for parallelization. It contains some modified code of the NAUTY / TRACES distribution (available at [2]), namely specialized versions of the Schreier-Sims implementation. The source code is available at [1].

The refinement of DEJAVU is a highly-engineered version of color refinement [17], following the implementation of TRACES closely. We also exploit the blueprint technique as described in [3], extending the technique to also cache cell selector choices for subsequent branches.

Next, we provide further details on practical and conceptual aspects of the solver.

4.1 Breadth-first Traversal and Trace Deviation Sets

The work of the breadth-first traversal is shared between threads through lock-free queues. A master thread adds all of the elements which have to be computed to a queue. Threads then dequeue a chunk of work, compute the elements, and report their results back to the master thread through another queue. In order to minimize overhead, large chunks of work are enqueued and dequeued from the queue rather than single elements. Furthermore, DEJAVU uses automorphism pruning when performing breadth-first search as described in Section 3.3 while filling up the queue (not enqueueing multiple elements known to be isomorphic).

During breadth-first traversal, we make use of a *trace* invariant, as introduced by TRACES and described in [17]. Furthermore, we introduce the *trace deviation set* technique. The approach is related to the special automorphism algorithm of TRACES [17] as well as the trace deviation trees used in [3].

During breadth-first traversal, we also keep a *trace deviation set*. The idea of this pruning technique is related to the special automorphism algorithm of TRACES (see [17]) and the trace deviation tree technique of [3]. We present the idea in the following.

To describe the technique, we first define a new node invariant, which we call the *deviation value* $\text{Dev}_{\text{Inv}}: V^* \rightarrow \mathbb{N}^2 \cup \{\perp\}$. Consider a fixed trace $\text{Inv}(\tau)$, which for our purposes will be the trace invariant of the target leaf τ . The deviation value $\text{Dev}_{\text{Inv}}(\nu)$ for a node ν is then defined as a tuple of the first position and the corresponding value in the trace $\text{Inv}(\nu)$ that is different from $\text{Inv}(\tau)$. If there are no differences, we set the deviation value to \perp denoting “no deviation”. Since the deviation value is a function of the invariant computed up until an isomorphism-invariant point, it is also naturally invariant under isomorphism.

Consider a node μ in the search tree. The crucial observation is that in our algorithm, we can also use the set of deviation values of its children as an invariant for μ itself. Assume ν_1, \dots, ν_k are children of μ and none of the subtrees rooted in the children has been pruned through $\text{Prune}_{\text{Inv}}$. Then, $D(\mu) := \{\text{Dev}_{\text{Inv}}(\nu_1), \dots, \text{Dev}_{\text{Inv}}(\nu_k)\}$, the trace deviation set of μ , can be used as an invariant for μ : we claim that for any other node μ' with children ν'_1, \dots, ν'_k , it must hold that $D(\mu) = \{\text{Dev}_{\text{Inv}}(\nu_1), \dots, \text{Dev}_{\text{Inv}}(\nu_k)\} = \{\text{Dev}_{\text{Inv}}(\nu'_1), \dots, \text{Dev}_{\text{Inv}}(\nu'_k)\} = D(\mu')$ whenever μ and μ' are isomorphic. If no pruning has taken place, this is easy to see since the branches are isomorphic by assumption, immediately implying that branches must contain the same invariant values. But if $\text{Prune}'_{\text{Aut}}$ is applied, no invariant values can be removed either: since pruned nodes are isomorphic to remaining nodes, they, again immediately by definition, must contain the same invariant values.

When advancing in a breadth-first manner, the aforementioned requirements are guaranteed to be satisfied: no $\text{Prune}_{\text{Inv}}$ has taken place on the level that is currently being pruned. Furthermore, while computing the level, the set of deviation values is automatically calculated anyway: whenever we observe that a node ν below μ deviates from the desired invariant $\text{Inv}(\nu)$ and should be pruned using $\text{Prune}_{\text{Inv}}$, we already have enough information to derive $\text{Dev}_{\text{Inv}}(\nu)$.

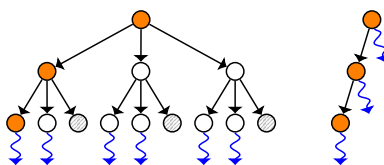
These observations are specifically exploited as follows: first, all children of the base node τ' (which belongs to the path on the way to the target leaf τ) are computed. If nodes deviate from the trace, their deviation values are recorded into a set, i.e., we calculate the trace deviation set $D(\tau')$. The idea is that if a node is (supposedly) isomorphic to the base node τ' , then, for its (supposedly) isomorphic children, it must deviate from the trace in the same manner at the same position. Hence, for all other parent nodes μ , we also keep track of $D(\mu)$ when calculating their children. Whenever we discover a new element of $D(\mu)$, we check whether the set equivalence $D(\tau') = D(\mu)$ can still be satisfied. If not, μ can be pruned immediately, without the necessity to calculate all its children.

For example, assume that we calculated deviation values $\{d_1, d_2\}$ for the base node (illustrated in Figure 1). From the previous discussion, it follows that we can immediately prune all nodes that produce a value other than $\{d_1, d_2\}$. We can also prune nodes that do not produce all of the deviation values. If for example a value $d_3 \notin \{d_1, d_2\}$ is encountered, the parent node can immediately be removed from the tree.

A crucial point is that pruning through trace deviation sets has negligible cost: children of the base node always have to be computed, and the trace deviation does not necessitate more calculation than is done for that particular node anyway. We are still able to fully use the early-out capabilities of the trace invariant. In the parallel implementation, only the initial recording of deviation values of base nodes has to be synchronized. In the implementation we do however, depending on a heuristic, use a slight variation: to make deviation values more distinct, it is sometimes beneficial to not use the early-out immediately. Instead, for a fixed constant k , color refinement is continued past the deviation for k more cells, accumulating more information for the deviation value. The trade-off is as follows: if k becomes larger, the early-out in color refinement is taken later, but deviation values become more distinct. In practice, this trades per-node cost for the number of nodes in the search tree. However, in our experiments we observed that even for very small k (we use $k = 5$), node reduction can be substantial – while not increasing per-node cost by a significant amount.

4.2 Automorphism Sampling

A central aspect when sampling automorphisms is whether we can guarantee that the resulting automorphisms are distributed uniformly in the group or not. If the distribution is uniform, they count towards the probabilistic abort criterion of the solver. The solver



■ **Figure 2** Level search (left) and base-aligned search (right). Squiggly lines indicate where random walks originate, orange indicates base nodes and gray nodes are pruned by breadth-first traversal.

uses two approaches, namely *base-aligned search* (generally non-uniform) and *level search* (uniform).

Level search is essentially sampling as described in Section 3 (see Figure 2, left). Walks are initiated from the remaining nodes of the search tree at the current breadth-first level, and when drawing the initial node, weights are accounted for to make the resulting distribution of automorphisms uniform. Crucially, in this mode, DEJAVU usually stores additional target leaves, which is proven to result in an exponential speedup in the worst-case [3].

Base-aligned search is designed to find as many automorphisms as possible with minimal effort. This typically entails giving up on uniformity. Base-aligned search initiates random walks from a base point of a given strategy (see Figure 2, right). Whenever finding automorphisms from a certain base point is deemed hard, search is advanced to the next base point. As a side effect, this handles inherently easy graphs efficiently: whenever color refinement already determines the orbit partition, base-aligned search finds all automorphisms and is even able to terminate search deterministically.

5 Practical Performance

We provide benchmark results corroborating that DEJAVU performs competitively on a large variety of graphs while scaling with the use of more threads.

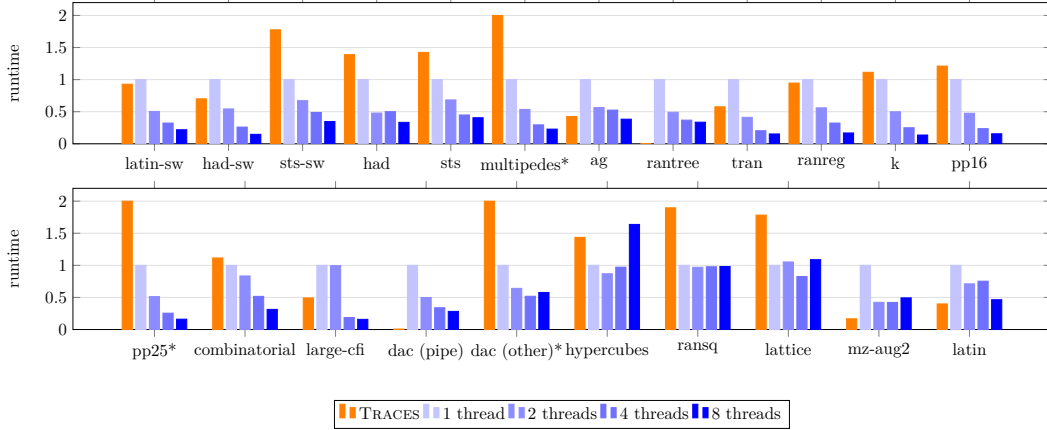
5.1 Benchmarks

All benchmarks were run on an Intel Core i7 9700K (8 cores) processor with 16GB of RAM and Ubuntu 19.04. All graphs were randomly permuted, but every solver was given the same permuted version of a graph. All runtimes are measured without parsing the input.

The benchmarks include most sets from [2], which is the de facto standard when it comes to symmetry computation. We extended two of the sets to larger instance sizes. The respective graphs can be found in [1]. We should point out that for random trees and pipe graphs TRACES benefits from specialized code that is not implemented in DEJAVU.

The (user-definable) error bound for DEJAVU was set to 1%. A 1% error bound means that with at most that probability at least 1 generator of the generating set is missing. It can be proven however that the probability of missing at least 2 generators then only has a probability of at most $\frac{1}{2}\%$, missing at least 3 only at most $\frac{1}{2^2}\%$ and so forth (the argument for this is similar to the proof of Lemma 8 involving the index of the subgroup found). Hence, even if errors occur, it is highly likely only a small part of the generating set is missing.

Actual error probabilities are even lower. Due to the one-sided nature of the error, on asymmetric graphs DEJAVU can not err (e.g. most random regular graphs, random graphs, multipedes, `latin-sw`). Secondly, on many graph families DEJAVU can invoke a deterministic criterion for termination (e.g. for most of `rantree`, `hypercubes`, `dac`, `lattice`, complete graphs, `tran`). This means for the majority of the benchmarks errors can not be observed.



■ **Figure 3** Relative runtimes of benchmark sets for TRACES and DEJAVU (using the respective number of threads). For sets marked with *, we capped the runtime of TRACES at 2.

The benchmarks corroborate that through the use of parallelism, we are able to achieve significant speedup over state-of-the-art tools in a domain that so far has been exclusively dominated by sequential solvers (see Figure 3). Specifically, we do so on the particular, representative benchmark suite state-of-the-art solvers are tuned to solve.

Overall benchmarks show that DEJAVU with a single thread performs competitively with TRACES (on 12 benchmark sets DEJAVU is faster, while TRACES is faster on the other 10 sets). Using 8 threads, DEJAVU outperforms TRACES on most sets (on 17 out of 22 sets). Additionally, on *lattice*, DEJAVU performs better than TRACES for larger instances.

5.2 Scaling

We want to discuss in detail how DEJAVU scales with the use of more threads. In Figure 3, scaling is illustrated. To lessen the bonus of sampling cell selectors (see Section 4) gained when adding threads, we provide the 1 and 2 thread variants with the best cell selector for each set. (Otherwise, scaling would be even better, but the comparison would not be fair.) The diagram shows the summed up runtimes for a graph class relative to the runtime of a single thread, i.e., the single thread variant has a runtime of 1. This might overrepresent larger instances in the data, but of course larger instances are exactly those of biggest interest.

The diagram shows that for most sets of the benchmark suite, on our hardware, DEJAVU scales remarkably with the use of more threads. Overall we achieved our goal of designing a competitive tool that on most graph classes can efficiently exploit parallelism, in our experiments sometimes even approaching what is theoretically possible on 8 cores.

6 Conclusion and Future Work

We presented the new randomized, parallel algorithm DEJAVU that computes automorphism groups and can be used to compute symmetries of combinatorial objects. Benchmarks show that DEJAVU is competitive with state-of-the-art solvers, and parallelizes to 8 cores remarkably well on a wide variety of instances. In order to show further scaling in the same generic manner, we believe developing more large-scale, meaningful benchmark sets is required. For harder instances, preliminary testing shows that the use of more cores significantly improves the runtimes even further.

In future work, we intend to improve DEJAVU by adding dedicated subroutines to handle low degree vertices and gadget graph constructions. On very simple graphs, color refinement is the bottleneck which seems to necessitate an efficient parallel implementation for color refinement.

References

- 1 dejavu. <https://www.mathematik.tu-darmstadt.de/dejavu>.
- 2 nauty and Traces. <http://pallini.di.uniroma1.it>.
- 3 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *2021 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2021. to appear. doi:10.1137/1.9781611976472.6.
- 4 Vijaya Balpande and Anjali Mahajan. Article: Parallelization of graph isomorphism using OpenMP. *International Journal of Computer Applications*, 117(8):33–41, May 2015. Full text available. doi:10.5120/20576-2982.
- 5 Duc-Hiep Chu and Joxan Jaffar. A complete method for symmetry reduction in safety verification. In *Proceedings of the 24th international conference on Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 616–633. Springer, 2012. doi:10.1007/978-3-642-31424-7_43.
- 6 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, Igor L. Markov, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference, DAC '04*, pages 530–534, New York, NY, USA, 2004. ACM. doi:10.1145/996566.996712.
- 7 Robert Gens and Pedro M. Domingos. Deep symmetry networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 2537–2545, 2014. URL: <https://dl.acm.org/doi/abs/10.5555/2969033.2969110>.
- 8 Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier, 2006. doi:10.1016/S1574-6526(06)80014-3.
- 9 Patrice Godefroid. Exploiting symmetry when model-checking software. In *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, volume 156 of *IFIP Conference Proceedings*, pages 257–275. Kluwer, 1999. doi:10.1007/978-0-387-35578-8_15.
- 10 Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, 2017. doi:10.1145/3107239.
- 11 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.13.
- 12 Hadi Katebi, Karem A. Sakallah, and Igor L. Markov. Symmetry and satisfiability: An update. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 113–127. Springer, 2010. doi:10.1007/978-3-642-14186-7_11.
- 13 Martin Kutz and Pascal Schweitzer. Screwbox: a randomized certifying graph-non-isomorphism algorithm. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.14.

- 14 Leo Liberti. Symmetry in mathematical programming. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 263–283, New York, NY, 2012. Springer New York. doi:10.1007/978-1-4614-1927-3_9.
- 15 Yanxi Liu, Hagit Hel-Or, Craig S. Kaplan, and Luc Van Gool. Computational symmetry in computer vision and computer graphics. *Foundations and Trends in Computer Graphics and Vision*, 5(1-2):1–195, 2010. doi:10.1561/9781601983657.
- 16 José Luis López-Presa, Luis Núñez Chiroque, and Antonio Fernández Anta. Novel techniques for automorphism group computation. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933 of *LNCS*, pages 296–307. Springer, 2013. doi:10.1007/978-3-642-38527-8_27.
- 17 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, 2014. doi:10.1016/j.jsc.2013.09.003.
- 18 Alice Miller, Alastair F. Donaldson, and Muffy Calder. Symmetry in temporal logic model checking. *ACM Computing Surveys*, 38(3), 2006. doi:10.1145/1132960.1132962.
- 19 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.
- 20 Irene Luque Ruiz and Miguel Ángel Gómez-Nieto. A Java tool for the management of chemical databases and similarity analysis based on molecular graphs isomorphism. In *Computational Science - ICCS 2008, 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part II*, volume 5102 of *LNCS*, pages 369–378. Springer, 2008. doi:10.1007/978-3-540-69387-1_41.
- 21 Ákos Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003. doi:10.1017/CB09780511546549.
- 22 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. doi:10.5555/1953048.2078187.
- 23 Greg Daniel Tener. *Attacks on Difficult Instances of Graph Isomorphism: Sequential and Parallel Algorithms*. PhD thesis, University of Central Florida, Orlando, FL, USA, 2009. AAI3401107.

Graph Connectivity and Single Element Recovery via Linear and OR Queries

Sepehr Assadi 

Rutgers University, New Brunswick, NJ, USA

Deeparnab Chakrabarty 

Dartmouth College, Hanover, NH, USA

Sanjeev Khanna 

University of Pennsylvania, Philadelphia, PA, USA

Abstract

We study the problem of finding a spanning forest in an undirected, n -vertex multi-graph under two basic query models. One are **Linear** queries which are linear measurements on the incidence vector induced by the edges; the other are the weaker **OR** queries which only reveal whether a given subset of plausible edges is empty or not. At the heart of our study lies a fundamental problem which we call the *single element recovery* problem: given a non-negative vector $x \in \mathbb{R}_{\geq 0}^N$, the objective is to return a single element $x_j > 0$ from the support. Queries can be made in rounds, and our goals is to understand the trade-offs between the query complexity and the rounds of adaptivity needed to solve these problems, for both deterministic and randomized algorithms. These questions have connections and ramifications to multiple areas such as sketching, streaming, graph reconstruction, and compressed sensing. Our main results are as follows:

- For the single element recovery problem, it is easy to obtain a deterministic, r -round algorithm which makes $(N^{1/r} - 1)$ -queries per-round. We prove that this is tight: any r -round *deterministic* algorithm must make $\geq (N^{1/r} - 1)$ **Linear** queries in some round. In contrast, a 1-round $O(\text{polylog}(N))$ -query *randomized* algorithm is known to exist.
- We design a *deterministic* $O(r)$ -round, $\tilde{O}(n^{1+1/r})$ -OR query algorithm for graph connectivity. We complement this with an $\tilde{\Omega}(n^{1+1/r})$ -lower bound for any r -round deterministic algorithm in the OR-model.
- We design a *randomized*, 2-round algorithm for the graph connectivity problem which makes $\tilde{O}(n)$ -OR queries. In contrast, we prove that any 1-round algorithm (possibly randomized) requires $\tilde{\Omega}(n^2)$ -OR queries. A *randomized*, 1-round algorithm making $\tilde{O}(n)$ -Linear queries is already known.

All our algorithms, in fact, work with more natural graph query models which are *special* cases of the above, and have been extensively studied in the literature. These are **Cross** queries (cut-queries) and **BIS** (bipartite independent set) queries.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Query Models, Graph Connectivity, Group Testing, Duality

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.7

Related Version *Full Version*: <https://arxiv.org/abs/2007.06098>

Funding *Sepehr Assadi*: Supported in part by NSF CAREER award CCF-2047061, and gift from Google Research.

Deeparnab Chakrabarty: Supported in part by the NSF awards CCF-1813053 and CCF-2041920.

Sanjeev Khanna: Supported in part by the NSF awards CCF-1763514, CCF-1617851, CCF-1934876, and CCF-2008305.



© Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 7; pp. 7:1–7:19



Leibniz International Proceedings in Informatics

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

1 Introduction

Many modern applications compel algorithm designers to rethink random access to input data, and revisit basic questions in a *query access model* where the input is accessed only via answers to certain kinds of queries. There are many reasons for this ranging from data volume (only snapshots of the data can be accessed) to data ownership (access is restricted via certain APIs).

In this paper, we study algorithms accessing an unknown, undirected multi-graph G on n vertices in the following two basic query models. Think of the graph as an unknown non-negative $\binom{n}{2}$ dimension vector x_G with $\text{supp}(x_G)$ denoting the positive coordinates. With this view, answers to these queries below can be interpreted as *measurements* on this vector.

- **Linear Queries (Linear):** Given any non-negative¹ $\binom{n}{2}$ dimension vector a_G , what is $a_G \cdot x_G$?
- **OR Queries (OR):** Given any subset S of the $\binom{n}{2}$ dimensions, is $\text{supp}(x_G) \cap S$ empty? Reverting back to the combinatorial nature of graphs, it is perhaps more natural to think of different kinds of queries, and indeed the following two have been extensively studied. These are however special² cases, respectively, of the queries above.
- **Cross-additive Queries (Cross):** Given two disjoint subsets A, B of V , $\text{Cross}(A, B)$ returns the number of edges, including multiplicity, that have one endpoint in A and the other in B .
- **Bipartite Independent Set Queries (BIS):** Given two disjoint subsets A, B of V , $\text{BIS}(A, B)$ returns whether or not there is an edge that has one endpoint in A and the other in B .

The above query models (and similar variants such as additive queries [30], cut-queries [49], edge-detection queries [6, 9]) have a rich literature [1, 4, 9, 16, 17, 30, 41, 44, 49]. Most previous works, however, have focused on either *graph reconstruction* [15–17, 41], or on *parameter estimation* (e.g., estimating the number of edges [9] or triangles [11]). In this work, however, our goal is to understand the power and limitations of these queries to reveal **structural properties** of the underlying graph. In particular, we study the following basic property.

► **Problem 1 (Graph connectivity).** *Given query access to an undirected multigraph on the vertex set $V = \{1, \dots, n\}$, return a spanning forest.*

It is not too hard to implement the *classic* BFS or DFS traversals to obtain an $\tilde{O}(n)$ -query deterministic algorithm for the above problem in either query model. However, such algorithms are *adaptive*, that is, the queries depend on the answers obtained so far. A much more modern algorithm of Ahn, Guha, and McGregor [2] gives³ an $\tilde{O}(n)$ -Linear query *non-adaptive* but *randomized* algorithm for the problem. This raises the following questions that motivate us

What is the rounds-of-adaptivity versus query-complexity trade-off for deterministic algorithms for Problem 1? Can randomization also help in the OR and BIS models?

It turns out that understanding the complexity of Problem 1 is closely related to understanding an even more basic problem which we discuss below.

¹ Non-negativity is for convenience. A general linear query can be broken into two non-negative queries.

² The Cross (and BIS queries) correspond to $\{0, 1\}$ vectors a_G (and subsets) corresponding to cuts. Indeed, our algorithms work with the weaker queries while our lower bounds will be for the stronger queries. It should also be clear that the Linear (and respectively Cross) queries are at least as strong as OR (resp. BIS) queries.

³ Using results in [51], one can also obtain a $\tilde{O}(n)$ -query deterministic algorithm in the Cross-query model.

Single Element Recovery

Consider a non-negative real-valued vector $x \in \mathbb{R}_{\geq 0}^N$ and suppose we have access to x only via **Linear** or **OR** queries where the dimension is now N . We define the following problem which we call the *single-element recovery* problem (following the standard “support-recovery” problem in compressed sensing).

► **Problem 2** (Single-element recovery). *Given a non-negative real-valued vector $x \in \mathbb{R}_{\geq 0}^N$, accessed via either **Linear**-queries or **OR**-queries, output any arbitrary element⁴ from the support $\text{supp}(x)$.*

To see how the above problem relates to Problem 1, consider the vector of possible edges incident to a single vertex. A spanning forest must find an edge incident to this vertex. This corresponds to solving Problem 2 on this vector. The problem is also interesting in its own right, with connections to *combinatorial group testing* [23, 24, 43], *compressed sensing* [19, 22, 33], and *coin-weighing problems*. [12, 30, 40, 50]. While most of these works have focused on recovering the full support, we ask the simpler question of just recovering a single element.

If one allows *randomization*, then one can use ℓ_0 -samplers [34] to solve the above problem using $O(\log^2 N \log(\frac{1}{\delta}))$ **Linear** queries⁵, *non-adaptively*. In fact, ℓ_0 -samplers return a random element in $\text{supp}(x)$. The parameter δ is the error probability. There have been numerous applications of these (see the table in Figure 1 of [37], for instance), and indeed many applications (including the AGM [2] algorithm alluded to above) need only an arbitrary element in the support. This is precisely what is asked in Problem 2. Furthermore, the upper bound for randomized algorithms is nearly tight [34, 37], and therefore, for randomized algorithms, our understanding is pretty much complete. But what can be said about *deterministically* finding a single support element⁶? This is an important question for it relates to deterministic analogs to the various applications stated above.

It is not too hard to make a couple of observations. One, any *non-adaptive* deterministic algorithm for Problem 2 using **Linear**-queries can in fact be recursively used to completely recover the whole vector. This implies an $\tilde{\Omega}(N)$ information theoretic lower bound. Two, if one allows more rounds, then one can indeed do better using a binary-search style idea. More precisely, in each round the algorithm partitions the search space into $N^{1/r}$ parts and using $N^{1/r}$ queries finds a non-zero part. In this way in r rounds, one can get algorithm making $N^{1/r}$ -queries per round. This leads to the following fundamental question which we answer in our paper.

What is the rounds-of-adaptivity versus query-complexity trade-off for deterministic algorithms for Problem 2?

1.1 Motivation and Perspective

Why should we care about the questions above?

- We think that algorithmic question of computation on graphs via queries is as natural and important as the reconstruction question. Indeed, our study was inspired by trying to understand the power of *cut*-queries to check whether a graph was connected or not; this

⁴ In the case of **OR**-queries, we can only return the j with $x_j > 0$

⁵ A similar result holds with **OR** queries as well. See Section 3

⁶ A “deterministic ℓ_0 sampler”, if you allow us the abuse of notation.

is an (extremely) special case of submodular function minimization. More recently, this type of “property-testing via queries” question on graphs has been asked for matchings by Nisan [44], and more generally for matrix properties by [51] and [46]. Single element recovery is also as natural as whole-vector recovery. Indeed, one can imagine a scenario where recovering a big⁷ subset of the support (diseased blood samples, say) faster and with fewer queries may be more beneficial than reconstructing the whole vector.

- The **Linear** query model is closely connected to *linear sketches* that have found plenty of applications in dynamic streaming; see, e.g. [29, 35, 39]. The single element recovery problem also has connections to the *universal relation* \mathbf{UR}^C problem in communication complexity, which was studied in [37, 42]. Understanding these questions, therefore, have ramifications to other areas. As a concrete example, one consequence of our results is a deterministic, $O(r)$ -pass dynamic streaming algorithm for graph connectivity in $\tilde{O}(n^{1+1/r})$ space. This was not known before.
- We believe the question of the trade-off between rounds versus query complexity is natural and important, especially in today’s world of massively parallel computing. Such trade-offs are closely related to similar questions in communication complexity, number of passes in streaming algorithms, etc. It is worthwhile building up an arsenal of tools to attack such questions. Indeed, one main contribution of this paper is to show how LP-duality can be used as one such tool.
- Why do we focus on deterministic algorithms? Mainly because, as mentioned above, our understanding of the complexity of randomized algorithms for the problems above is near complete. However, in some applications one may require exponentially low error, or has to deal with an “adversary” (say, the one giving updates to a streaming algorithm) that is not oblivious to the algorithm’s randomness; see, e.g. [10]. This further motivates the study of deterministic algorithms in this context. Furthermore, we need to design lower-bounding techniques which only work against deterministic algorithms, and this is of technical interest.

1.2 Our Results

Our first result is a tight lower bound for the question on single element recovery. The binary-search style algorithm mentioned above is the best one can do.

► **Result 1.** *For the single element recovery with **Linear**-query access, any r -round, deterministic algorithm must make $\geq N^{1/r} - 1$ queries in some round.*

We should remind the reader that the above lower bound is for vectors whose domain is non-negative rationals. In particular, it does not hold for Boolean vectors⁸. Moving to the continuous domain allows one to use tools from geometry, in particular duality theory and Caratheodory’s theorem, to prove the tight lower bound.

As mentioned above, **Linear** queries are stronger than **OR** queries, and thus the above lower bound holds for **OR** queries as well. The proof for **OR** queries, however, is combinatorial, arguably simpler, and more importantly can be generalized to prove the following lower bound for Problem 1 as well.

⁷ As we show later in Lemma 16, algorithmically we can get results when the “single” in single element recovery can be larger.

⁸ Indeed, for Boolean vector with **Linear** queries one can recover the whole vector if the query vector has exponentially large coefficients. Even when the coefficients are small ($\{0, 1\}$ even), the vector can be recovered with $O(n/\log n)$ -queries which is information theoretically optimal.

► **Result 2.** *Any r -round deterministic algorithm for finding a spanning forest, must make $\tilde{\Omega}(n^{1+\frac{1}{r}})$ -OR queries.*

As we explain below, the above smooth trade-off between rounds and query complexity is optimal, even when we allow the weaker BIS-queries. Algorithmically, we have the following result. We mention that such a result was not known even using Linear or Cross queries. A similar lower bound as in Result 2 with Cross-queries is left open.

► **Result 3.** *For any positive integer r , there exists an $O(r)$ -round deterministic algorithm which makes $\tilde{O}(n^{1+\frac{1}{r}})$ -BIS queries per round, and returns a spanning forest of the graph.*

It is worth remarking that our algorithm with Linear queries (which is implied by the weaker BIS queries) above also implies an $O(r)$ -pass $\tilde{O}(n^{1+1/r})$ -space *deterministic* algorithm for maintaining a spanning forest in *dynamic* graph streams. As the edge updates arise, one simply updates the answers to the various queries made in each round. This result was not known before.

Finally, we show that for Problem 1, randomization is helpful in decreasing the number of rounds. More precisely, we consider Monte-Carlo algorithms.

► **Result 4.** *There exists a 2-round randomized algorithm for graph connectivity which makes $\tilde{O}(n)$ -OR queries per round. There exists a 4-round randomized algorithm for graph connectivity which makes $\tilde{O}(n)$ -BIS queries per round. Any non-adaptive, randomized algorithm for graph connectivity must make $\tilde{\Omega}(n^2)$ -OR queries.*

Table 1 summarizes our contributions.

■ **Table 1** Summary of the state-of-the-art and our results for graph connectivity and single-element recovery problems. In each cell, we write the number of rounds followed by the query complexity per round. All lower bounds are with respect to the stronger model (Linear and OR). For upper bounds, if there is a discrepancy between the stronger and weaker models, we show this using a | as partition. Bold results are ours. The remaining results are folklore unless a reference is explicitly cited. The ? indicates the main open question of our paper.

		Linear Cross		OR BIS	
		Upper Bound	Lower Bound	Upper Bound	Lower Bound
Single Element Recovery	Det	$r, N^{1/r} - 1$	$r, N^{1/r} - 1$	$r, N^{1/r} - 1$	$r, N^{1/r} - 1$
	Rand	$r = 1, O(\log^2 N)$	$r = 1, \Omega(\log^2 N)$ [34]	$r = 1, O(\log^2 N)$	$r = 1, \Omega(\log^2 N)$
Graph Connectivity	Det	$O(r), n^{1+1/r}$?	$O(r), n^{1+1/r}$	$r, \tilde{\Omega}(n^{1+1/r})$
	Rand	$r = 1, \tilde{O}(n)$ [2] $\tilde{O}(n)$ [2, 51]	$r, \Omega(n/\log n)$	$r = 2 4, \tilde{O}(n)$	$r = 1, \tilde{\Omega}(n^2)$

1.3 Technical Overview

In this section we give a technical overview of our results. These highlight the main underlying ideas and will assist in reading the detailed proofs which appear in the subsequent sections.

Overview of Result 1. It is relatively easy to prove an r -round lower bound for single element recovery in the OR-query model via an adversary argument (see [7]). At a high level, OR-queries only mildly interact with each other and can be easily fooled. Linear queries, on the other hand, strongly interact with each other. To illustrate: if we know $x(A)$ and $x(B)$ for $B \subseteq A$, then we immediately know $x(A \setminus B)$. This is untrue for OR-queries – if x has a non-zero entry in both A and B , nothing can be inferred about its entries in $A \setminus B$. Indeed, this power manifests itself in the non-adaptive, randomized algorithm using Cross-queries; it is important that we can use subtraction. This makes proving lower bounds against Linear-queries distinctly harder.

In our proof of Result 1, we use duality theory. To highlight our idea, for simplicity, let's consider a warmup *non-adaptive* problem. The algorithm has to ask $\ll \sqrt{N}$ queries, and on obtaining the response, needs to return a subset $S \subseteq [N]$ of size $\ll \sqrt{N}$ with the guarantee that $\text{supp}(x) \cap S$ is not empty. Note that if this were possible, then there would be a simple 2-round $o(\sqrt{N})$ -algorithm – simply query the individual coordinates of S in the second round. This is what we want to disprove. Therefore, given the first round's $\ll \sqrt{N}$ queries, we need to show there exists responses such that *no matter* which set S of $\ll \sqrt{N}$ size is picked, there exists a feasible $x \in \mathbb{R}_{\geq 0}^N$ which sets all entries in S to 0. Note this is a $\exists \forall \exists$ -statement. How does one go ahead establishing this?

We first observe that for a fixed response \mathbf{a} and a fixed set S , whether or not a feasible $x \in \mathbb{R}_{\geq 0}^N$ exists is asking whether a system of linear inequalities has a feasible solution. Farkas Lemma, or taking the dual, tells us exactly when this is the case. The nice thing about the dual formulation is that the “response” \mathbf{a} becomes a “variable” in the dual program, as it should be since we are trying to find it. To say it another way, taking the dual allows us to assert conditions that the response vector \mathbf{a} must satisfy, and the goal becomes to hunt for such a vector. How does one do that? Well, the conditions are once again linear inequalities, and we again use duality. In particular, we use Farkas Lemma again to obtain conditions certifying the *non-existence* of such an \mathbf{a} . The final step is showing that the existence of this certificate is impossible. This step uses another tool from geometry – Carathéodory's theorem. Basically, it shows that if a certificate exists, then a *sparse* certificate must exist. And then a simple counting argument shows the impossibility of sparse certificates. This, of course, is an extremely high-level view and for just the warmup problem. In Section 2 we give details of this warmup, and also details of how one proves the general r -round lower bound building on it.

The interested reader may be wondering about the two instantiations of duality (isn't the dual of the dual the primal?). We point out that duality can be thought of as transforming a \exists statement into a \forall statement: feasibility is a \exists statement, Farkas implies infeasibility is a different \exists statement, and negating we get the original feasibility as a \forall statement. Since we were trying to assert a $\exists \forall \exists$ -statement, the two instantiations of duality hit the two different \exists .

Overview of Result 2. At a high level, the lower bound for Problem 1, the spanning forest problem, boils down to a “direct sum” version of Problem 2, the single element recovery problem. Imagine the graph is an $n \times n$ bipartite graph. Therefore, finding a spanning forest requires us finding an edge incident to *each* of the n vertices on one side. This is precisely solving n -independent versions of Problem 2 in parallel. However, note that a single query can “hit” different instances at once. The question is, as all direct-sum questions are, does this make the problem n -times harder? We do not know the answer for Linear queries and leave this as the main open question of our work. However, we can show that the simpler,

combinatorial proof of Result 1 against OR-queries does have a direct-sum version, and gives an almost tight lower bound for Problem 1. This is possible because OR-queries, as mentioned in the previous paragraph, have only mild interaction between them. We show that this interaction cannot help by more than a $\text{poly}(r)$ -factor. Our proof is an adversary argument, and a similar argument was used recently by Nisan [44] to show that matchings cannot be approximated well by deterministic algorithms with OR-queries. Details of this are given in [7].

Overview of Result 3. In Section 3, we show some simple, folklore, and known results for single element recovery. We build on these algorithms to obtain our algorithms for Problem 1. With every vertex one associates an unknown vector which is an indicator of its neighborhood. If one applies the r -round binary-search algorithm for the single element recovery problem on each such vector, then in r -rounds with $O(n^{1+1/r})$ -BIS queries, for every vertex one can obtain a single edge incident on it. This alone however doesn't immediately help: perhaps, we only detect $n/2$ edges and get $n/2$ disconnected clusters. Recursively proceeding only gives an $O(r \log n)$ -round algorithm. And we would like no dependence on n .

To make progress, we actually give a more sophisticated algorithm for single element recovery than binary search, which gives more and may be of independent interest. In particular, we describe an algorithm (Lemma 16) for single element recovery which in $O(r)$ rounds, and making $N^{1/r}$ -queries per round, can in fact return as many as $N^{1/4r}$ elements in the support. Once we have this, then for graph connectivity we observe that in $O(r)$ rounds, we get polynomially many edges incident on each vertex. Thus as rounds go on, the number of effective vertices decreases, which allows us to query more aggressively. Altogether, we get an $O(r)$ -round algorithm making only $\tilde{O}(n^{1+1/r})$ -BIS queries. The details of this are described in Section 4.

Overview of Result 4. In the overview of the deterministic algorithm, we had to be a bit conservative in that even after every vertex found k edges (k being 1 or $n^{O(1/r)}$) incident on it, we pessimistically assumed that after this step the resulting graph still has $\Theta(n/k)$ disconnected clusters, and we haven't learned *anything* about the edges across these clusters. In particular, we allow for the situation that the cross-cluster edges can be dense. With randomization, however, we get to sample k *random* edges incident on a vertex. This is where we use the recent result of Holm *et al.* [31] which shows that if the k incident edges are random, then, as long as $k = \Omega(\log n)$, the number of *inter-component* edges between the connected components induced by the sampled edges, is $O(n/k)$. That is the cross-cluster edges are sparse. Therefore, in a single round with $\tilde{O}(n)$ -randomized BIS queries, we can obtain a disconnected random subgraph, but one such that, whp, there exist at most $\tilde{O}(n)$ edges across the disconnected components.

Given the above fact, the algorithm is almost immediate. After round 1, we are in a sparse graph (where nodes now correspond to subsets of already connected vertices). If we were allowed general OR-queries, then a single round with $\tilde{O}(n)$ -OR queries suffices to learn this sparse graph, which in turn, gives us a spanning forest in the original graph. This follows from algorithms for single element recovery when the vector is promised to be sparse (discussed in Section 3). Unfortunately, these queries may not be BIS-queries; recall that BIS-queries are restricted to ask about edges across two subsets. Nevertheless, we can show how to implement the above idea using 2-extra rounds with only BIS-queries, giving a 4-round algorithm. Details can be found in [7].

To complement the above, we also prove that even with randomization, one cannot get non-adaptive (1-round) $o(n^2/\log^2 n)$ -query algorithms with OR-queries. Indeed, the family of examples is formed by two cliques (dense graphs) which could have a single edge, or not, that connects them. A single collection of $o(n^2/\log^2 n)$ -OR queries cannot distinguish between these two families. Details can be found in [7].

1.4 Related Works

Our work falls in the broad class of algorithm design in the *query access model*, where one has limited access to the input. Over the years there has been a significant amount of work relevant to this paper including in graph reconstruction [1, 3, 4, 6, 12, 14, 15, 17, 30, 41, 47], parameter estimation [9, 11, 21, 48], minimum cuts [8, 49] sketching and streaming [2, 5, 8, 27, 28, 34, 36, 37, 42, 51], combinatorial group testing, compressed sensing, and coin weighing [12, 19, 22–25, 50]. It is impossible to do complete justice, but in the full version [7] we give a more detailed discussion of some of these works and how they fit in with our paper.

2 Lower Bound for Single Element Recovery

The following is the formal statement of the r -round lower bound for single element recovery.

► **Theorem 1.** *Any r -round deterministic algorithm for Single Element Recovery must make $\geq (N^{1/r} - 1)$ -Linear queries in some round.*

In fact, we prove (see Corollary 9) that if k_1, \dots, k_r are r positive integers such that $\prod_{i=1}^r (k_i + 1) < N$, then no r -round algorithm making $\leq k_i$ queries in round i can be successful for the Single Element Recovery problem. This implies Theorem 1. To begin with, we give a proof for essentially the $r = 2$ case, which was the warm-up question we discussed in Section 1.3. More precisely, we prove that if $(k + 1)s < N$ then no one-round algorithm making $\leq k$ queries can return a subset S of size $\leq s$ with $x_j > 0$ for some $j \in S$. That is, a subset which traps an element in $\text{supp}(x)$. This essentially implies the $r = 2$ case with $k_1 = k$ and $k_2 = s - 1$. This proof contains the core ideas behind the more general statement, which follows via an inductive application of the same idea. The complete proof of Theorem 1 can be found in Section 2.1. For now, we focus on proving the following statement.

► **Theorem 2.** *If $(k + 1)s < N$, then there cannot exist a 1-round deterministic algorithm making k -Linear queries for the trapping problem with parameter s .*

Note that if s divides N , then $k = \frac{N}{s} - 1$ queries suffice and so the above theorem is tight.

Proof. Since x is a non-zero, non-negative vector, by scaling, we assume that $x([N]) = 1$. We let \mathbf{A} denote the $k \times N$ matrix corresponding to the k queries arranged as row vectors. We use $\mathbf{a} \in \mathbb{R}_{\geq 0}^k$ to denote the answers we will give to fool any algorithm. To find this, fix any subset S with $|S| \leq s$, and consider the following system of inequalities parametrized by the answer vector \mathbf{a} . The only inequalities are the non-negativity constraints. Below, and throughout, $[N] := \{1, 2, \dots, N\}$.

$$\mathcal{P}(\mathbf{a}; S) = \{x \in \mathbb{R}_{\geq 0}^N : x([N]) = 1 \quad \mathbf{A} \cdot x = \mathbf{a} \quad x(S) = 0\} \quad (\text{P})$$

Note that if $\mathcal{P}(\mathbf{a}; S)$ has a feasible solution, then given the answers \mathbf{a} to its queries, the algorithms *cannot* return the subset S . This is because there is a non-negative x consistent with these answers with S disjoint from its support. In other words, S is *safe* for the lower

bound w.r.t. \mathbf{a} . Therefore, if there exists an answer vector \mathbf{a} such that *every* subset $S \subseteq [N]$ with $|S| \leq s$ is safe with respect to \mathbf{a} , that is $\mathcal{P}(\mathbf{a}; S)$ is feasible, then we would have proved our lower bound. We use duality and geometry to prove the existence of this vector (if $(k+1)s < N$).

The first step is to understand when for a *fixed* set S , the system $\mathcal{P}(\mathbf{a}; S)$ is infeasible. This is answered by Farkas Lemma. In particular, consider the following system⁹ of inequalities where the variables are the Lagrange multipliers corresponding to the equalities in (P). We note that the variables are *free*, since $\mathcal{P}(\mathbf{a}; S)$ has only equalities in the constraints. For convenience, we have eliminated the variable corresponding to the subset S and have moved it to the right hand side. Below, and throughout, for any subset $S \subseteq [N]$, we use $\mathbf{1}_S$ to denote the N -dimensional 0,1-vector which has 1 in the coordinates $i \in S$.

$$\mathcal{C}_S := \left\{ (y^{(0)}, \mathbf{y}) \in \mathbb{R} \times \mathbb{R}^k : y^{(0)} \cdot \mathbf{1}_{[N]} + \mathbf{y} \cdot \mathbf{A} \leq \mathbf{1}_S \right\}$$

Farkas Lemma asserts that $\mathcal{P}(\mathbf{a}; S)$ is *infeasible* if and only if there exists $(y^{(0)}, \mathbf{y}) \in \mathcal{C}_S$ such that $y^{(0)} \cdot \mathbf{1} + \mathbf{y} \cdot \mathbf{a} > 0$. Contrapositively, we get that $\mathcal{P}(\mathbf{a}; S)$ is feasible, that is S is safe with respect to \mathbf{a} , iff $y^{(0)} + \mathbf{y} \cdot \mathbf{a} \leq 0$ for *all* $\mathbf{y} \in \mathcal{C}_S$. Since we want an answer \mathbf{a} such that *all* subsets S with $|S| \leq s$ are safe, we conclude that such an answer exists if and only if the following system of linear inequalities has a feasible solution.

$$\mathcal{Q} := \left\{ \mathbf{a} \in \mathbb{R}_{\geq 0}^k : \mathbf{y} \cdot \mathbf{a} \leq -y^{(0)}, \quad \forall S \subseteq [N], |S| \leq s, \quad \forall (y^{(0)}, \mathbf{y}) \in \mathcal{C}_S \right\} \quad (\text{D})$$

In summary, to prove the lower bound, it suffices to show that \mathcal{Q} has a feasible solution, and this solution will correspond to the answers to the queries. Suppose, for the sake of contradiction, \mathcal{Q} is *infeasible*. Then, again by Farkas Lemma (but on a different system of inequalities), there exists multipliers $\lambda_t > 0$ corresponding to constraints $(S_t \text{ s.t. } |S_t| \leq s, (y_t^{(0)}, \mathbf{y}_t) \in \mathcal{C}_{S_t})$ for some $t = 1 \dots T$ such that (P1) $\sum_{t=1}^T \lambda_t \mathbf{y}_t \geq \mathbf{0}_k$ where $\mathbf{0}_k$ is the k -dimensional all zero (row) vector, and (P2) $\sum_{t=1}^T \lambda_t y_t^{(0)} > 0$. Note that this time λ_t 's are non-negative since \mathcal{Q} has inequalities in the constraints.

We can focus on the λ_t 's which are *positive* and discard the rest. The next key observation is to *upper bound* the size T of the support. Note that the conditions (P1) and (P2) can be equivalently stated as asserting that the $(k+1)$ -dimensional *cone* spanned by the vectors $(y_t^{(0)}, \mathbf{y}_t)$ contains a non-negative point with first coordinate positive. Caratheodory's theorem (for cones) asserts that any such point can be expressed as a conic combination of at most $(k+1)$ vectors. Therefore, we can assume that $T \leq k+1$.

Now we are almost done. Since $(y_t^{(0)}, \mathbf{y}_t) \in \mathcal{C}_{S_t}$, we have $y_t^{(0)} \cdot \mathbf{1}_{[N]} + \mathbf{y}_t \cdot \mathbf{A} \leq \mathbf{1}_{S_t}$. Taking λ_t combinations and adding, we get (since all $\lambda_t > 0$) that

$$\left(\sum_{t=1}^T \lambda_t y_t^{(0)} \right) \cdot \mathbf{1}_{[N]} + \left(\sum_{t=1}^T \lambda_t \mathbf{y}_t \right) \cdot \mathbf{A} \leq \sum_{t=1}^T \lambda_t \mathbf{1}_{S_t}$$

Since every $|S_t| \leq s$, the support of the right hand side vector is $\leq sT \leq s(k+1)$. The support of the left hand side vector is $= N$. This is because the second summation is a non-negative vector by (P1), and the first has full support. This contradicts $(k+1)s < N$. Hence, \mathcal{Q} has a feasible solution, which in turn means there exists answers \mathbf{a} which foils \mathbf{A} . This proves Theorem 2. \blacktriangleleft

⁹ Here \mathbf{y} and $\mathbf{1}_S$ are *row vectors*. In the general proof, there will be multiple \mathbf{y} 's indexed with super-scripts. All of them are row-vectors.

2.1 The General r -round Lower Bound

We begin by formally defining what an r -round deterministic algorithm is, and what it means for such an algorithm to successfully solve Single Element Recovery.

► **Definition 3** (r -round deterministic algorithm.). An r -round deterministic algorithm \mathcal{A} proceeds by making a collection of linear queries $\mathbf{A}^{(1)} \in \mathbb{R}_{\geq 0}^{k_1 \times N}$ and obtains the answer $\mathbf{a}^{(1)} = \mathbf{A}^{(1)} \cdot x$. This is the first round of the algorithm. For $1 < i \leq r$, in the i th round the algorithm makes a collection of linear queries $\mathbf{A}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i \times N}$. This matrix depends on the history $(\mathbf{A}^{(1)}, \mathbf{a}^{(1)}), \dots, (\mathbf{A}^{(i-1)}, \mathbf{a}^{(i-1)})$. Upon making this query it obtains the answer $\mathbf{a}^{(i)} = \mathbf{A}^{(i)} \cdot x$. We call $\Pi_r := ((\mathbf{A}^{(1)}, \mathbf{a}^{(1)}), \dots, (\mathbf{A}^{(r)}, \mathbf{a}^{(r)}))$ the r -round **transcript** of the algorithm. The output of the deterministic algorithm \mathcal{A} only depends on Π_r .

A vector $y \in \mathbb{R}_{\geq 0}^N$ is said to be **consistent** with respect to a transcript Π_r if $\mathbf{A}^{(i)} \cdot y = \mathbf{a}^{(i)}$ for all $1 \leq i \leq r$. A transcript $\Pi_r = ((\mathbf{A}^{(1)}, \mathbf{a}^{(1)}), \dots, (\mathbf{A}^{(r)}, \mathbf{a}^{(r)}))$ is **feasible** for the algorithm if there is some vector y consistent with respect to it, and if the algorithm indeed queries $\mathbf{A}^{(i)}$ given the $(i-1)$ -round transcript $((\mathbf{A}^{(1)}, \mathbf{a}^{(1)}), \dots, (\mathbf{A}^{(i-1)}, \mathbf{a}^{(i-1)}))$.

► **Definition 4.** An r -round deterministic algorithm \mathcal{A} is said to successfully solve Single Element Recovery if for all non-zero $x \in \mathbb{R}_{\geq 0}^N$, upon completion of r -rounds the algorithm \mathcal{A} returns a coordinate $j \in [N]$ with $x_j > 0$. In particular, if the algorithm returns a coordinate j given a feasible transcript Π_r , then every x that is consistent with Π_r must have $x_j > 0$.

For technical reasons, we add a 0th-round for any r -round algorithm. In this round, the query “matrix” $\mathbf{A}^{(0)}$ is the single N -dimensional row with all ones. That is, we ask for the sum of x_j for all $j \in [N]$. We assume that the answer $\mathbf{a}^{(0)}$ is the scalar 1 to capture the fact that the vector x is non-zero.

Next we define the notion of safe subsets with respect to a transcript generated till round i . A safe subset of coordinates are those for which there is a consistent vector x whose support is disjoint from the subset, that is, $x_j = 0$ for all $j \in S$, or equivalently $x(S) = 0$ since $x \geq 0$.

► **Definition 5.** Given an i -round transcript $\Pi_i = ((\mathbf{A}^{(0)}, \mathbf{a}^{(0)}), \dots, (\mathbf{A}^{(i)}, \mathbf{a}^{(i)}))$, a subset $S \subseteq [N]$ is **safe** w.r.t. Π_i if the following system of linear inequalities

$$\mathcal{P}(\mathbf{a}^{(\leq i)}; S) := \left\{ x \in \mathbb{R}^N : \begin{cases} \mathbf{A}^{(j)} \cdot x = \mathbf{a}^{(j)} & \forall 0 \leq j \leq i \\ x(S) = 0 \\ x \geq 0 \end{cases} \right\} \quad (\text{Primal})$$

has a feasible solution.

► **Claim 6.** If Π_r is a feasible r -round transcript of an algorithm \mathcal{A} such that all singletons are safe w.r.t Π_r , then the algorithm \mathcal{A} cannot be successful in solving Single Element Recovery.

Proof. Given Π_r , the algorithm \mathcal{A} must return some coordinate $j \in [N]$. However $\{j\}$ is safe. That is, there is a feasible solution x to $\mathcal{P}(\mathbf{a}^{(\leq r)}, \{j\})$. Indeed, if x were the input vector, the algorithm would return a coordinate not in the support. \triangleleft

► **Definition 7** (Transcript Creation Procedure). Given an r -round algorithm \mathcal{A} , the transcript creation procedure is the following iterative process. In round i , given the transcript $\Pi_{i-1} := ((\mathbf{A}^{(0)}, \mathbf{a}^{(0)}), \dots, (\mathbf{A}^{(i-1)}, \mathbf{a}^{(i-1)}))$ upon which the algorithm \mathcal{A} queries $\mathbf{A}^{(i)}$, and the transcript creation procedure produces an answer $\mathbf{a}^{(i)}$ such that $\Pi_i = \Pi_{i-1} \circ (\mathbf{A}^{(i)}, \mathbf{a}^{(i)})$ is feasible.

Our main theorem, which implies Theorem 1, is the following.

► **Theorem 8** (Transcript Creation Theorem). *Let k_1, \dots, k_r and s_0, s_1, \dots, s_r be positive integers such that $s_0 \leq n - 1$ and $(k_i + 1)s_i \leq s_{i-1}$ for all $i \geq 1$. Then given any r -round algorithm \mathcal{A} making $\leq k_i$ queries in round i , there is a transcript creation procedure to create an r -round transcript such that for all $0 \leq i \leq r$, any subset $S \subseteq [N]$ with $|S| \leq s_i$ is safe with respect to Π_i .*

► **Corollary 9.** *Let k_1, \dots, k_r be any r positive integers with $\prod_{i=1}^r (k_i + 1) < n$. No r -round algorithm \mathcal{A} which makes $\leq k_i$ queries in round i can be successful for the Single Element Recovery problem. In particular, this implies Theorem 1.*

Proof. Set $s_r = 1$, $s_{r-1} = (k_r + 1)$, and in general, $s_i = (k_r + 1)(k_{r-1} + 1) \cdots (k_{i+1} + 1)$. Note that the conditions of Theorem 8 are satisfied. Therefore given any algorithm \mathcal{A} making $\leq k_i$ queries in round i , we can create a r -round transcript such that all singleton sets are safe with respect to Π_r . Claim 6 implies \mathcal{A} cannot be successful. ◀

2.1.1 Proof of the Transcript Creation Theorem

We start with writing the dual representation of safe sets. Fix a subset $S \subseteq [N]$ and a transcript Π_i . By Farkas lemma we know that the system $\mathcal{P}(\mathbf{a}^{(\leq i)}; S)$ is infeasible only if there exists a infeasibility certificate

$$\left(\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \dots, \mathbf{y}^{(i)} \right) \in \mathbb{R} \times \mathbb{R}^{k_1} \times \dots \times \mathbb{R}^{k_i} : \sum_{j=0}^i \mathbf{y}^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \quad \text{and} \quad \sum_{j=0}^i \mathbf{y}^{(j)} \cdot \mathbf{a}^{(j)} > 0$$

Here $\mathbf{1}_S$ is the n -dimensional indicator vector of the subset S , that is, it has 1 in the coordinates $j \in S$ and 0 otherwise. Taking negations, we get that the system $\mathcal{P}(\mathbf{a}^{(\leq i)}; S)$ is *feasible*, that is $S \subseteq [N]$ is safe w.r.t Π_{i-1} , if and only if the following condition holds

$$S \text{ is safe w.r.t. } \Pi_i \text{ iff } \mathbf{y}^{(\leq i)} \cdot \mathbf{a}^{(\leq i)} := \sum_{j=0}^i \mathbf{y}^{(j)} \cdot \mathbf{a}^{(j)} \leq 0 \quad \text{for all } \mathbf{y}^{(\leq i)} \in \mathcal{C}_S^{(i)} \quad (\text{Dual})$$

where,

$$\mathcal{C}_S^{(i)} := \left\{ \mathbf{y}^{(\leq i)} := (\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \dots, \mathbf{y}^{(i)}) \in \mathbb{R} \times \mathbb{R}^{k_1} \times \dots \times \mathbb{R}^{k_i} : \sum_{j=0}^i \mathbf{y}^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \right\}$$

We are now ready to prove Theorem 8 via induction on i . The above representation is the dual definition of safe sets, and this definition is what is easy to induct with.

Base Case: $i = 0$. We need to show that any subset $S \subseteq [N]$ of size $|S| \leq s_0 = N - 1$ is safe with respect to the transcript $(\mathbf{A}^{(0)}, \mathbf{a}^{(0)})$. To remind the reader, $\mathbf{A}^{(0)}$ is just the all ones vector and $\mathbf{a}^{(0)}$ is just the scalar 1. Using (Dual), we need to show for any subset $S \subseteq [N]$ with $|S| \leq N - 1$, we must have

$$\mathbf{y}^{(0)} \cdot \mathbf{a}^{(0)} \leq 0 \quad \text{for all } \mathbf{y}^{(0)} \in \mathbb{R} \text{ such that } \mathbf{y}^{(0)} \cdot \mathbf{A}^{(0)} \leq \mathbf{1}_S$$

However, $\mathbf{y}^{(0)} \cdot \mathbf{A}^{(0)}$ is the n -dimensional vector which is $\mathbf{y}^{(0)}$ on all coordinates. Since $|S| \leq n - 1$, there is some coordinate $j \notin S$ such that $\mathbf{1}_S[j] = 0$. Thus, $\mathbf{y}^{(0)} \leq 0$ implying $\mathbf{y}^{(0)} \cdot \mathbf{a}^{(0)} \leq 0$. The base case holds.

Inductive Case: $i \geq 1$. Assume the conclusion of the theorem holds for all $0 \leq j \leq i-1$. That is, there is a procedure which has created a transcript $\Pi_{i-1} = ((\mathbf{A}^{(0)}, \mathbf{a}^{(0)}), \dots, (\mathbf{A}^{(i-1)}, \mathbf{a}^{(i-1)}))$ such that every subset $S \subseteq [N]$ with $|S| \leq s_{i-1}$ is safe w.r.t Π_{i-1} . Using (Dual), we can rewrite this as the following statement

$$\forall S \subseteq [N], |S| \leq s_{i-1}, \quad \text{for all } \mathbf{y}^{(\leq i-1)} \in \mathcal{C}_S^{(i-1)} \quad \text{we have} \quad \mathbf{y}^{(\leq i-1)} \cdot \mathbf{a}^{(\leq i-1)} \leq 0. \quad (\text{IH})$$

Given Π_{i-1} , the algorithm \mathcal{A} now queries $\mathbf{A}^{(i)}$ in round i . Our goal is to find answers $\mathbf{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i}$ such that any subset $S \subseteq [N]$ with $|S| \leq s_i$ is safe w.r.t $\Pi_i = \Pi_{i-1} \circ (\mathbf{A}^{(i)}, \mathbf{a}^{(i)})$. Again referring to (Dual), we need to find $\mathbf{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i}$ satisfying the following system of linear inequalities.

$$\mathcal{Q}^{(i)} := \left\{ \mathbf{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i} : \mathbf{y}^{(i)} \cdot \mathbf{a}^{(i)} \leq -(\mathbf{y}^{(\leq i-1)} \cdot \mathbf{a}^{(\leq i-1)}), \quad \forall S \subseteq [N], |S| \leq s_i, \quad \forall \mathbf{y}^{(\leq i)} \in \mathcal{C}_S^{(i)} \right\}$$

Although it may appear that the above system has infinitely many constraints, it suffices to write the constraints for extreme points for the polyhedra $\mathcal{C}_S^{(i)}$'s. To complete the proof, we need to show that $\mathcal{Q}^{(i)}$ is non-empty; if so, we can select any $\mathbf{a}^{(i)} \in \mathcal{Q}^{(i)}$ for completing the transcript creation procedure, and proving the theorem by induction. The next lemma does precisely that; this completes the proof of the theorem. ◀ Theorem 8

► **Lemma 10.** *The system of inequalities $\mathcal{Q}^{(i)}$ has a feasible solution.*

Proof. For the sake of contradiction, suppose not. Applying Farkas lemma (again), we get the following certificate of infeasibility. There exists the tuples $(\lambda_t > 0, S_t \subseteq [N])$ with $|S_t| \leq s_i$, $\mathbf{y}_t^{(\leq i)} \in \mathcal{C}_{S_t}^{(i)}$ for $1 \leq t \leq k_i + 1$ such that

(P1): $\sum_{t=1}^{k_i+1} \lambda_t \mathbf{y}_t^{(i)} \geq \mathbf{0}_{k_i}$, and

(P2): $\sum_{t=1}^{k_i+1} \lambda_t (\mathbf{y}_t^{(\leq i-1)} \cdot \mathbf{a}^{(\leq i-1)}) > 0$.

Since $\mathbf{y}_t^{(\leq i)} \in \mathcal{C}_{S_t}^{(i)}$, we get $\sum_{j=0}^i \mathbf{y}_t^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_{S_t}$ for all $1 \leq t \leq k_i$. Taking the positive λ_t -combinations of these inequalities, we get

$$\sum_{t=1}^{k_i+1} \lambda_t \cdot \left(\sum_{j=0}^i \mathbf{y}_t^{(j)} \cdot \mathbf{A}^{(j)} \right) \leq \sum_{t=1}^{k_i+1} \lambda_t \mathbf{1}_{S_t} \quad (\text{P3})$$

Now, define $\mathbf{w}^{(j)} := \sum_{t=1}^{k_i+1} \lambda_t \mathbf{y}_t^{(j)}$ for $0 \leq j \leq i$. (P1) above implies (Q1): $\mathbf{w}^{(i)} \geq \mathbf{0}_{k_i}$, and (P2) implies (Q2): $\mathbf{w}^{(\leq i-1)} \cdot \mathbf{a}^{(\leq i-1)} > 0$. And finally, (P3) translates to

$$\underbrace{\sum_{j=0}^{i-1} \mathbf{w}^{(j)} \cdot \mathbf{A}^{(j)}}_{\text{Call this } \mathbf{u}_1} + \underbrace{\mathbf{w}^{(i)} \cdot \mathbf{A}^{(i)}}_{\text{Call this } \mathbf{u}_2} \leq \underbrace{\sum_{t=1}^{k_i+1} \lambda_t \mathbf{1}_{S_t}}_{\text{Call this } \mathbf{v}} \quad (\text{Q3})$$

Now we are ready to see the contradiction. First observe that the vector \mathbf{v} has at most $(k_i + 1)s_i$ positive entries since it is the sum of $k_i + 1$ vectors each of support $\leq s_i$. Since $\mathbf{w}^{(i)}$ and $\mathbf{A}^{(i)}$ are both non-negative, \mathbf{u}_2 is a non-negative vector. This implies that \mathbf{u}_1 must have $\leq (k_i + 1)s_i$ positive entries. From the conditions of the theorem, we get $(k_i + 1)s_i \leq s_{i-1}$. Thus, \mathbf{u}_1 has $\leq s_{i-1}$ positive entries. This in turn implies there exists a scalar θ such that $\theta \mathbf{u}_1 \leq \mathbf{1}_S$ for some subset $S \subseteq [N]$ with $|S| \leq s_{i-1}$. That is,

$$\sum_{j=0}^{i-1} (\theta \mathbf{w}^{(j)}) \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \quad \Rightarrow \quad \theta \mathbf{w}^{(\leq i-1)} \in \mathcal{C}_S^{(i-1)}$$

The induction hypothesis (IH) implies $\theta \mathbf{w}^{(\leq i-1)} \cdot \mathbf{a}^{(\leq i-1)} \leq 0$. This contradicts (Q2). This completes the proof of the lemma. ◀

3 Algorithms Warmup: Algorithms for Single Element Recovery

In this section, we state some simple and/or well known algorithms for single element recovery which we use these as subroutines for our algorithms for graph connectivity as well.

► **Lemma 11.** *Let $x \in \mathbb{R}_{\geq 0}^N$ be a non-zero, non-negative vector, and let r be a positive integer. There exists an r -round deterministic algorithm $\text{BinarySearch}_r(x)$ which makes $(N^{1/r} - 1)$ -OR queries per round, and returns a coordinate j with $x_j > 0$. If Linear queries are allowed, then one can recover x_j as well.*

Proof. (Sketch) Divide $[N]$ into $N^{1/r}$ blocks each of size $N^{1-1/r}$, and run OR query on each block but the last, taking $N^{1/r} - 1$ queries in all. If one of them evaluates to 1, recurse on that for the next $r - 1$ rounds. Otherwise, recurse on the last block. ◀

Next, we state a standard result from the combinatorial group testing and coin-weighing literature [19, 25, 32, 38, 45] which says that if the support of x is known to be small, then there exist efficient *one-round* deterministic algorithms to recover the complete vector.

► **Lemma 12.** [32, 45] *Let $x \in \mathbb{R}_{\geq 0}^N$ be a non-zero vector, and let d be any positive integer. There exists a 1-round (non-adaptive) deterministic algorithm $\text{BndSuppRec}(x, d)$ which makes $O(d^2 \log N)$ -OR queries and (a) either asserts $\text{supp}(x) > d$, or (b) recovers the full support of x . With Linear queries, the number of queries reduces to $O(d \log N)$.*

Proof. (Sketch) We give a very high level sketch only for the sake of completeness. For the case of $d = 1$, take the $O(\lceil \log N \rceil \times N)$ matrix A where column i is the number i represented in binary. Then Ax (the “OR product”) points to the unique element in the support. To see the *existence* of a deterministic procedure for larger d , one can proceed by the probabilistic method. If one samples each coordinate with probability $1/d$, then with constant probability the vector restricted to this sample has precisely support 1 for which the above “ $d = 1$ ” algorithm can be used to recover it. Repeating this $O(d \log N)$ times leads to error probability which swamps the union bound over $\leq N^d$ possible sets, implying the existence of a deterministic scheme. Finally, another $O(d)$ arises since we need to recover all the $\leq d$ coordinates. All this can be made explicit by using ideas from error correcting codes; we point the interested reader to [32, 45] for the details. ◀

Next we move to randomized algorithms. Here ideas from F_0 -estimation [5, 27] and ℓ_0 -sampling [18, 28, 34] give the following algorithms.

► **Lemma 13.** *Let $x \in \mathbb{R}_{\geq 0}^N$ be a non-zero vector. There exists a 1-round (non-adaptive) randomized algorithm $\text{RandSuppSamp}(x)$ which makes $O(\log^2 N \log(\frac{1}{\delta}))$ -OR queries and returns a random $j \in \text{supp}(x)$ with probability $\geq 1 - \delta$.*

Proof. (Sketch) Suppose we knew the support $\text{supp}(x) = d$. Then, we sample each $j \in [N]$ with probability $1/d$ to get a subset $R \subseteq [N]$. With constant probability $\text{supp}(x \cap R) = 1$ and, conditioned on that, it contains a random $j \in \text{supp}(x)$. Therefore, running the algorithm $\text{BndSuppRec}(x \cap R, 1)$ asserted in Lemma 12, we can find a random $j \in \text{supp}(x)$ with constant probability. Repeating this $O(\log(1/\delta))$ times gives the desired error probability. Since we don’t know $\text{supp}(x)$, we run for various powers of 2 in 1 to N . ◀

► **Lemma 14.** (Theorem 7 in [13], also in [20, 26]) *Let $x \in \mathbb{R}_{\geq 0}^N$ be a non-zero vector. There exists a 1-round (non-adaptive) randomized algorithm $\text{SuppEst}(x)$ which makes $O(\log N \cdot \log(1/\delta))$ -OR-queries and returns an estimate \tilde{s} of the support which satisfies $\frac{\text{supp}(x)}{3} \leq \tilde{s} \leq 3\text{supp}(x)$ with probability $\geq 1 - \delta$.*

4 Deterministic Algorithm for Graph Connectivity

In this section, we prove the following theorem which formalizes Result 3.

► **Theorem 15.** *Let r be any fixed positive integer. There exists an $35r$ -round deterministic algorithm $\text{DetGraphConn}(G)$ which makes at most $O(n^{1+\frac{1}{r}} \log n)$ -BIS-queries per round on an undirected multigraph G , and returns a spanning forest of G .*

We start by establishing some simple subroutines which we need. We will refer to some algorithmic paradigms defined in Section 3.

4.1 Simple Subroutines

We begin by strengthening the simple algorithm BinarySearch asserted in Lemma 11. While in r -rounds with $O(N^{1/r})$ -OR queries $\text{BinarySearch}_r(x)$ recovers a single element in the support, one can in fact get many more elements from the support. This result may be of independent interest.

► **Lemma 16.** *Let $x \in \mathbb{R}_+^N$ be a non-zero, non-negative vector, and let r be a positive integer, and let $c < r$. There exists a $\lceil 2r/c \rceil$ -round deterministic algorithm $\text{DetFindMany}_{r,c}(x)$ which makes $O(N^{c/r} \log N)$ -OR queries per round, and returns $\min(N^{c/4r}, \text{supp}(x))$ distinct coordinates from $\text{supp}(x)$.*

Proof. In the first round, we partition the range $[N]$ into $N^{c/2r}$ blocks of size $N^{1-c/2r}$ each. Let these blocks be B_1, \dots, B_k with $k = N^{c/2r}$. For each $i \in [k]$, we run the algorithm $\text{BndSuppRec}(x \cap B_i, N^{c/4r})$ asserted in Lemma 12. The total number of queries used here is $O(N^{c/2r} \cdot (N^{c/4r})^2 \log N) = O(N^{c/r} \log N)$.

At the end of this round, either we recover $\text{supp}(x \cap B_i)$ for each block, and thus recover $\text{supp}(x)$, and we are done. Or, there is at least one block of size $N^{1-c/2r}$ which is guaranteed to contain $\geq N^{c/4r}$ elements in its support. We call this the heavy block of round 1. Next, we now proceed to recover $N^{c/4r}$ elements from this heavy block of round 1.

In the second round, we partition the indices of this heavy block again into $N^{c/2r}$ blocks of size $N^{1-2c/2r}$ each, and run BndSuppRec again on this block with $d = N^{c/4r}$. Once again, either we recover the entire support of the heavy block (which is guaranteed to contain at least $N^{c/4r}$ elements) and we are done. Or find a block of size $N^{1-2c/2r}$ that contains at least $N^{c/4r}$ elements in its support—this is the heavy block of round 2—and we now proceed to recover $N^{c/4r}$ elements in the heavy block of round 2.

We continue in this manner, and after $\lceil 2r/c \rceil - 1$ rounds, either we have already recovered at least $N^{c/4r}$ elements in the support of x , or have identified a heavy block of size $N^{1-(\lceil \frac{2r}{c} \rceil - 1) \cdot \frac{c}{2r}} = N^{c/2r}$ that contains at least $N^{c/4r}$ elements in the support of x . In the final round, we can simply probe each entry completing the proof. ◀

► **Remark 17.** The trade-off between the number of queries and number of elements recovered is not tightly established for the purpose of what we need in the graph connectivity algorithm. For instance, using the same idea as above, in 2 rounds one can actually recover $\min(N^{1/4}, \text{supp}(x))$ coordinates making $O(N^{3/4})$ -queries per round.

Next, we give an algorithm to find edges between two disjoint sets of vertices using BIS-queries.

► **Lemma 18.** *Let A and B be two disjoint sets of vertices with at least one edge between them. There exists a $2r$ -round deterministic algorithm $\text{DetFindEdge}_r(A, B)$ which makes $O(|A|^{1/r} + |B|^{1/r})$ -BIS queries per round, and returns an edge (a, b) with $a \in A$ and $b \in B$.*

Proof. Consider the $|B|$ dimensional vector x where x_b indicates the number of edges from a vertex $b \in B$ to vertices in A . We can simulate an OR-query in this vector using a BIS-query in the graph – for any subset $S \subseteq B$, $\text{OR}(S)$ on x has the same answer as $\text{BIS}(A, S)$. Therefore, using Lemma 11, in r -rounds and $|B|^{1/r}$ -BIS-queries, we can find a coordinate $b^* \in B$ with $x_{b^*} > 0$. That is, there is an edge between b^* and some vertex in A .

We can find one such vertex $a \in A$ to which b^* has an edge, again as above. We define the $|A|$ -dimensional vector y where y_a indicates the number of edges from b^* to a . Once again, the OR-query on y can be simulated using a BIS query on the graph – for any subset $S \subseteq A$, $\text{OR}(S)$ on y is the same as $\text{BIS}(S, \{b^*\})$. ◀

4.2 The Connectivity Algorithm

Now we give the $O(r)$ -round deterministic algorithm to find a spanning forest. First, we need the following simple claim.

▷ **Claim 19.** Let $G(V, E)$ be an arbitrary connected multigraph graph on n vertices, and let D be an arbitrary integer in $\{0, 1, \dots, (n-1)\}$. Let V_L denote all vertices in V which has at most D neighbors in G , and let $V_H = V \setminus V_L$. Let $E' \subseteq E$ be an arbitrary set of edges that satisfies the following property: for each vertex $u \in V_L$, the set E' contains all edges incident on u , and for every each vertex $v \in V_H$, the set E' contains D arbitrary edges incident on v . Then the graph $G' = (V, E')$ contains at most $\lfloor n/D \rfloor$ connected components.

Proof. Suppose G' has $K \geq \frac{n}{D}$ connected components. Thus, there must exist some component C with $\leq D$ vertices. Firstly, that C must have some vertex $v \in V_H$. If not, then since vertices in V_L have all their edges in G also in G' , this component would be disconnected in G which contradicts G 's connectedness. Secondly, observe that this leads to a contradiction: v has at least D neighbors in G' , and since there are at most $D-1$ other vertices in C , one of v 's neighbor in G' must lie outside C . This contradicts that C is a connected component. ◀

We are now ready to describe the algorithm $\text{DetGraphConn}(G)$. For simplicity, assume G is connected and our goal is to find a spanning *tree*. Subsequently, we explain how to modify the algorithm to find a spanning forest of a general graph. The algorithm proceeds in $O(\log r)$ phases starting with phase 0. The input to phase i is a partition $\Pi_i = (S_1, \dots, S_p)$ of the vertices. Each S_j in Π_i is guaranteed to be a connected in the graph G . Π_0 is the trivial partition of n singletons. Given Π_i , we define the graph $G_i = (\Pi_i, \mathcal{E}_i)$ where \mathcal{E}_i is the collection of *pseudo-edges* between components: we have a pseudo-edge $(S_a, S_b) \in \mathcal{E}_i$ if and only if there exists some edge in G between a vertex $u \in S_a$ and a vertex $v \in S_b$. Thus, G_0 is indeed the original graph. Note that by our assumption that G is connected, all the G_i 's are connected. We will be collecting pseudo-edges which will imply the connected components; we initialize this set \mathcal{F} to empty set. We will maintain the following invariant for a phase: $|\Pi_i| \leq n^{1-\frac{4^i-1}{r}}$; this is certainly true for $i = 0$. Next, we describe a phase i .

1. For each $S \in \Pi_i$, we construct a vector x indexed by all sets in $\Pi_i \setminus S$ where x_T indicates whether there is a pseudo-edge (S, T) in G_i . Next, we run the algorithm $\text{DetFindMany}_{r,c}(x)$ asserted in Lemma 16 to either find all pseudo-edges incident on S , or at least $n^{4^i/4r}$ of them. To do so, we set c such that $N^{c/4r} = n^{4^i/4r}$, where N is the dimension of x . That is, $N = |\Pi_i| - 1$. Indeed, we should set $c = \theta \cdot 4^i$ where $N^\theta = n$. Note, $\theta \geq 1$. Also note that the OR-queries on x can be simulated using BIS-queries on the original graph G . This is because we are looking at edges between S and a union of a subset of parts in $\Pi_i \setminus S$.

The number of rounds is $\lceil \frac{2r}{c} \rceil \leq \lceil \frac{2r}{4^i} \rceil$. The number of BIS-queries per round is $O(N^{c/r} \log N) = O(n^{4^i/r} \log n)$ per subset $S \in \Pi_i$. And thus, the total number of queries made is $N \cdot O(n^{4^i/r} \log n) \leq |\Pi_i| \cdot O(n^{4^i/r} \log n) \leq n^{1-\frac{4^i-1}{r}} \cdot O(n^{4^i/r} \log n) = O(n^{1+1/r} \cdot \log n)$.

2. Let $\mathcal{E}'_i \subseteq \mathcal{E}_i$ be the pseudo-edges obtained from the previous step. Let \mathcal{F}'_i be an arbitrary spanning forest of \mathcal{E}'_i . We add all these edges to the collection \mathcal{F} . Note, \mathcal{F}'_i is a collection of $\leq |\Pi_i|$ pseudo-edges.
3. Applying Claim 19 to the graph G_i , adding the pseudo edges in \mathcal{E}'_i reduces the number of connected components to at most $|\Pi_i|/n^{\frac{4^i}{4r}}$. We now repeat the above two steps 11 more times *sequentially*, and each time the number of connected components multiplicatively drops by $n^{\frac{4^i}{4r}}$. Thus, after the 12 sub-phases we end up with the partition Π_{i+1} of connected components, with $|\Pi_{i+1}| \leq |\Pi_i|/n^{\frac{12 \cdot 4^i}{4r}} \leq n^{1-\frac{4^i-1}{r}} \cdot n^{-\frac{12 \cdot 4^i}{4r}} = n^{1-\frac{4^{i+1}-1}{r}}$, as desired. The second inequality follows from the invariant before phase $(i+1)$ started.

To summarize, Phase i performs $O(\frac{r}{4^i})$ -rounds and makes $O(n^{1+1/r} \log n)$ -BIS queries per round. We run phase 0 to $L = O(\log r)$, till we get $|\Pi_L| \leq \sqrt{n}$. After that we run a clean up phase.

4. *Clean-up Phase.* Once $|\Pi_L| = O(\sqrt{n})$, for each pair (S, T) in $\Pi_L \times \Pi_L$, we make a single BIS-query to detect if the pseudo-edge $(S, T) \in \mathcal{E}_L$. The total number of queries is $O(n)$. We add an arbitrary spanning tree of \mathcal{E}_L to the set \mathcal{F} . At this point, \mathcal{F} lets us know the structure of connectivity via pseudo-edges. The next step is to recover the actual graph edges.
5. *Tree Building Phase.* Note that the total number of pseudo-edges in \mathcal{F} is $< n - 1$. For each $(S, T) \in \mathcal{F}$, we now desire to find an edge (s, t) in the graph where $s \in S$ and $t \in T$. Note that once we do this, we have the spanning tree the graph. This can be done in $2r$ more rounds using the algorithm $\text{DetFindEdge}_r(S, T)$ using $O(|S|^{1/r} + |T|^{1/r})$ -BIS queries per round. Therefore, the total number of queries per round of this phase is $O(n) \cdot O(n^{1/r}) = O(n^{1+1/r})$.

The number of rounds is $\sum_{i=1}^{O(\log r)} \frac{24r}{4^i} + 1 + 2r \leq 35r$.

This ends the description of the algorithm when G is connected. If G had more than one connected component, then one can recognize the connected components as the algorithm progresses. More precisely, if the algorithm is processing the partition $\Pi_i = (S_1, \dots, S_p)$ and find that S_i has no edges coming out of it, then by the invariant that S_i is connected, the algorithm can discard this component and proceed on the remaining graph as if it were connected. The analysis becomes better as the effective number of vertices decrease but the number of available queries don't. This completes the proof of Theorem 15.

References

- 1 Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. *CoRR*, abs/1803.10639, 2018.
- 2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc., SODA*, pages 459–467, 2012.
- 3 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics (SIDMA)*, 18(4):697–712, 2005.
- 4 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. In *Proc., FOCS*, page 197, 2002.
- 5 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. System Sci.*, 58(1):137–147, 1999.

- 6 Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *J. Comput. System Sci.*, 74(4):546–556, 2008.
- 7 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. *arXiv preprint arXiv:2007.06098*, 2020.
- 8 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proc., STOC*, pages 265–276, 2019.
- 9 Paul Beame, Sarel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. on Algorithms (TALG)*, 16(4):1–27, 2020. Preliminary version in *Proc. ITCS*, 2018.
- 10 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proc., ACM Symposium on Principles of Database Systems (PODS)*, 2020.
- 11 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using polylogarithmic queries. *CoRR*, abs/1808.00691, 2018.
- 12 Nader H Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *Proc., Conf. on Learning Theory*, 2009.
- 13 Nader H Bshouty. Lower bound for non-adaptive estimation of the number of defective items. In *Proc., International Symposium on Algorithms and Computation (ISAAC 2019)*, pages 2:1–2:9, 2019.
- 14 Nader H Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theoretical Computer Science*, 412(19):1782–1790, 2011.
- 15 Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theoretical Computer Science*, 417:23–35, 2012.
- 16 Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In *Proc., Conf. on Learning Theory*, pages 797–818, 2013.
- 17 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proc., STOC*, pages 749–758, 2008.
- 18 Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0 -sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.
- 19 Graham Cormode and S Muthukrishnan. Combinatorial algorithms for compressed sensing. In *SIROCCO*, pages 280–294. Springer, 2006.
- 20 Peter Damaschke and Azam Sheikh Muhammad. Competitive group testing and learning hidden vertex covers with minimum adaptivity. *Discrete Mathematics, Algorithms and Applications*, 2(03):291–311, 2010.
- 21 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In *Proc., STOC*, pages 281–288. ACM, 2018.
- 22 David L Donoho et al. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.
- 23 Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.
- 24 Dingzhu Du and Frank K Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.
- 25 AG D’yachkov, VS Lebedev, PA Vilenkin, and SM Yekhanin. Cover-free families and super-imposed codes: constructions, bounds and applications to cryptography and group testing. In *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No. 01CH37252)*, page 117. IEEE, 2001.
- 26 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Estimating the number of defectives with group testing. In *Proc., IEEE International Symposium on Information Theory (ISIT)*, pages 1376–1380, 2016.
- 27 Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. System Sci.*, 31(2):182–209, 1985.


- 28 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18:3–28, 2008.
- 29 Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *International Computer Science Symposium in Russia (CSR)*, pages 204–215, 2008.
- 30 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.
- 31 Jacob Holm, Valerie King, Mikkel Thorup, Or Zamir, and Uri Zwick. Random k-out subgraph leaves only $O(n/k)$ inter-component edges. In *Proc., FOCS*, pages 896–909. IEEE, 2019.
- 32 FK Hwang and VT Sós. Non-adaptive hypergeometric group testing. *Studia Sci. Math. Hungar.*, 22(1-4):257–263, 1987.
- 33 Piotr Indyk. Explicit constructions for compressed sensing of sparse signals. In *Proc., SODA*, pages 30–33, 2008.
- 34 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for ℓ_p samplers, finding duplicates in streams, and related problems. In *Proc., ACM Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.
- 35 John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In *Proc., STOC*, pages 1223–1236, 2020.
- 36 Michael Kapralov, Yin Tat Lee, CN Musco, Christopher Paul Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing (SICOMP)*, 46(1):456–477, 2017.
- 37 Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *Proc., FOCS*, pages 475–486, 2017.
- 38 W Kautz and Roy Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964.
- 39 Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proc., STOC*, pages 174–183, 2014.
- 40 Bernt Lindström. On Möbius functions and a problem in combinatorial number theory. *Canadian Mathematical Bulletin*, 14(4):513–516, 1971.
- 41 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *Proc., SODA*, pages 608–615, 2010.
- 42 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proc., SODA*, pages 1844–1860, 2019.
- 43 Hung Q Ngo and Ding-Zhu Du. A survey on combinatorial group testing algorithms with applications to dna library screening. *Discrete mathematical problems with medical applications*, 55:171–182, 2000.
- 44 Noam Nisan. The demand query model for bipartite matching. *Proc., SODA*, pages 592–599, 2021.
- 45 Ely Porat and Amir Rothschild. Explicit non-adaptive combinatorial group testing schemes. In *Proc., ICALP*, pages 748–759, 2008.
- 46 Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In *Proc., International Workshop on Randomization and Computation (RANDOM)*, pages 26:1–26:20, 2020.
- 47 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, pages 285–297. Springer, 2007.
- 48 Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Transactions on Computation Theory (TOCT)*, 8(4):15, 2016.
- 49 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *Proc., Innovations in Theoretical Computer Science (ITCS)*, pages 39:1–39:16, 2018.

- 50 Miklós Ruzinkó and Peter Vanroose. How an Erdős-Rényi-type search approach gives an explicit code construction of rate 1 for random access with multiplicity feedback. *IEEE Transactions on Information Theory*, 43(1):368–373, 1997.
- 51 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In *Proc., ICALP*, pages 94:1–94:16, 2019.

Fully Dynamic Set Cover via Hypergraph Maximal Matching: An Optimal Approximation Through a Local Approach

Sepehr Assadi 

Rutgers University, New Brunswick, NJ, USA

Shay Solomon 

Tel Aviv University, Israel

Abstract

In the (fully) dynamic set cover problem, we have a collection of m sets from a universe of size n that undergo element insertions and deletions; the goal is to maintain an approximate set cover of the universe after each update. We give an $O(f^2)$ update time algorithm for this problem that achieves an f -approximation, where f is the maximum number of sets that an element belongs to; under the unique games conjecture, this approximation is best possible for any fixed f . This is the first algorithm for dynamic set cover with approximation ratio that *exactly* matches f (as opposed to *almost* f in prior work), as well as the first one with runtime *independent of* n, m (for any approximation factor of $o(f^3)$).

Prior to our work, the state-of-the-art algorithms for this problem were $O(f^2)$ update time algorithms of Gupta et al. [STOC'17] and Bhattacharya et al. [IPCO'17] with $O(f^3)$ approximation, and the recent algorithm of Bhattacharya et al. [FOCS'19] with $O(f \cdot \log n / \varepsilon^2)$ update time and $(1 + \varepsilon) \cdot f$ approximation, improving the $O(f^2 \cdot \log n / \varepsilon^5)$ bound of Abboud et al. [STOC'19].

The key technical ingredient of our work is an algorithm for maintaining a *maximal* matching in a dynamic hypergraph of rank r – where each hyperedge has at most r vertices – that undergoes hyperedge insertions and deletions in $O(r^2)$ amortized update time; our algorithm is randomized, and the bound on the update time holds in expectation and with high probability. This result generalizes the maximal matching algorithm of Solomon [FOCS'16] with constant update time in ordinary graphs to hypergraphs, and is of independent merit; the previous state-of-the-art algorithms for set cover do not translate to (integral) matchings for hypergraphs, let alone a maximal one. Our quantitative result for the set cover problem is translated directly from this qualitative result for maximal matching using standard reductions.

An important advantage of our approach over the previous ones for approximation $(1 + \varepsilon) \cdot f$ (by Abboud et al. [STOC'19] and Bhattacharya et al. [FOCS'19]) is that it is inherently *local* and can thus be distributed efficiently to achieve low amortized round and message complexities.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases dynamic graph algorithms, hypergraph, maximal matching, matching, set cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.8

Related Version *Full Version*: <https://arxiv.org/pdf/2105.06889.pdf>

Funding *Sepehr Assadi*: Research supported in part by a NSF CAREER Grant CCF-2047061, and a gift from Google Research.

Shay Solomon: Partially supported by the Israel Science Foundation grant No.1991/19.



© Sepehr Assadi and Shay Solomon;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 8; pp. 8:1–8:18



Leibniz International Proceedings in Informatics

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

1 Introduction

In the set cover problem, we are given a family $\mathcal{S} = (S_1, \dots, S_m)$ of m sets on a universe $[n]$. The goal is to find a minimum-size subfamily of sets $\mathcal{F} \subseteq \mathcal{S}$ such that \mathcal{F} covers all the elements of $[n]$. Throughout the paper, we use f to denote the maximum frequency of any element $i \in [n]$ inside the sets in \mathcal{S} (by frequency of element i , we mean the number of sets in \mathcal{S} that contain i).

The set cover problem is one of the most fundamental and well-studied NP-hard problems, with two classic approximation algorithms, both with runtime $O(fn)$: greedy $\ln n$ -approximation and primal-dual f -approximation. One cannot achieve approximation $(1 - \varepsilon) \ln n$ unless $P = NP$ [26, 17] or approximation $f - \varepsilon$ for any fixed f under the unique games conjecture [20].

In recent years there is a growing body of work on this problem in the dynamic setting, where one would like to efficiently maintain a set cover for a universe that is subject to element insertions and deletions. The holy grail is to coincide with the bounds of the static setting: approximation factor of either $\ln n$ or f with (amortized) update time $O(f)$ (as a static runtime of $\Theta(fn)$ means that we spend $\Theta(f)$ time per each element of the universe on average). Indeed, in any dynamic model where element updates are explicit, update time $\Omega(f)$ is inevitable. Even in stronger models where updates are supported implicitly in constant time, recent SETH-based conditional lower bounds imply that update time $O(f^{1-\varepsilon})$ requires polynomial approximation factor [1].

The dynamic set cover problem was first studied by Gupta et al. [18] who gave an $O(\log n)$ -approximation with update time $O(f \log n)$ based on a greedy algorithm. The rest of the known algorithms are primal-dual-based and their approximation factor depends only on f . The state-of-the-art algorithms are: (1) An $O(f^3)$ -approximation with $O(f^2)$ update time, by Gupta et al. [18] and independently by Bhattacharya et al. [9], and (2) A $(1 + \varepsilon) \cdot f$ -approximation with update time $O(f \cdot \log n / \varepsilon^2)$ by Bhattacharya et al. [11], improving the $O(f^2 \cdot \log n / \varepsilon^5)$ bound of Abboud et al. [1] as well as an earlier result by Bhattacharya et al. [10]. Interestingly, the state-of-the-art algorithms for this problem are all *deterministic* (the algorithm of [1] is randomized however).

Our Result

In this work we demonstrate the power of randomization for the dynamic set cover problem by achieving the best possible approximation of f with runtime independent of both m, n :

► **Theorem (Informal).** *There is an algorithm for the dynamic set cover problem that achieves an exact f -approximation in $O(f^2)$ expected (and with high probability) amortized update time.*

This gives the first algorithm for dynamic set cover with approximation ratio that *exactly* matches f (as opposed to *almost* f in prior work), as well as the first one with runtime *independent of n, m* (for any approximation factor of $o(f^3)$). The bound $O(f^2)$ on the update time of our algorithm holds with high probability and in expectation. As in [1] and in the great majority of randomized dynamic graph algorithms, we assume an oblivious adversary. We shall remark that even for the much simpler problem of dynamic vertex cover (corresponding to $f = 2$ case), no algorithm is known against an adaptive adversary that achieves an exact 2-approximation in time independent of other input parameters (although $(2 + \varepsilon)$ -approximation algorithms have been known for some time now [13, 9, 6, 8]).

Maximal Hypergraph Matching

The key technical ingredient of our work is an algorithm for maintaining a *maximal* matching in a dynamic hypergraph of rank r – where each hyperedge has at most r vertices – that undergoes hyperedge insertions and deletions in $O(r^2)$ amortized update time. This result generalizes the maximal matching algorithm of Solomon [25] with constant update time for ordinary graphs, and is of independent merit; the previous state-of-the-art algorithms for set cover do not translate to (integral) matchings for hypergraphs, let alone a maximal one (however the algorithm of [1] translates to an integral (non-maximal) matching). The result for set cover follows immediately from this: taking all the matched vertices in a maximal hypergraph matching of rank f yields an f -approximate hypergraph vertex cover, or equivalently, an f -approximate set cover; (see Section 2 for details of this standard reduction).

We stress that there is an inherent difference between a maximal matching (yielding exactly f -approximation) and an almost-maximal matching (yielding $(1 + \varepsilon) \cdot f$ -approximation), wherein an ε -fraction of the potentially matched vertices may be unmatched; this is true in general but particularly important in the dynamic setting. Despite extensive work on dynamic algorithms for matching and vertex cover in ordinary graphs ($f = 2$), the state-of-the-art deterministic algorithm (or even randomized against adaptive adversary) for 2-approximate matching and vertex cover (via a maximal matching) has update time $O(\sqrt{|E|})$ [21], while $(2 + \varepsilon)$ -approximate matching and vertex cover can be maintained deterministically in poly-log update time [6, 7]; for approximate vertex cover and *fractional* matching, the update time can be further reduced to $O(1/\varepsilon^2)$ [13]. Therefore, it is only natural that our algorithm, which achieves an integral maximal hypergraph matching, is randomized and assumes an oblivious adversary.

Distributed networks

There is a growing body of work on distributed networks that change dynamically (cf. [22, 24, 15, 3, 19, 14, 2]). A distributed network can be modeled as an undirected (hyper)graph $G(V, E)$, where each vertex $v \in V$ is identified with a unique processor and the edge set E corresponds to direct communication links between the processors. In a static distributed setting all processors wake up simultaneously, and computation proceeds in fault-free synchronous rounds during which every processor exchanges messages of size $O(\log n)$ with its neighbors. We consider the standard *CONGEST* model (cf. [23]), which captures the essence of spatial locality and congestion. For hypergraphs, the messages should be of size $O(\log m)$, where m is the number of edges in the graph.

We focus on the standard setting in dynamic graph algorithms – dynamically changing networks that undergo edge updates (both insertions and deletions, a single edge update per step), which initially contain no edges. But in a *distributed* network we are subject to the following *local* constraint: After each edge update, only the affected vertices – the endpoints of the updated edge – are woken up; this is referred to in previous work as the (*CONGEST*) *local wakeup model*. Those affected vertices initiate an update procedure, which also involves waking up the vertices in the network (beyond the affected ones) that are required to participate in the update procedure, to adjust all outputs to agree on a valid global solution – in our case a maximal hypergraph matching; the output of each vertex is the set of its incident edges that belong to the matching. We make a standard assumption that the edge updates occur in large enough time gaps, and hence the network is always “stable” before the next change occurs (see, e.g., [22, 15, 3]). In this setting, the goal is to optimize (1) the number of communication rounds, and (2) the number of messages, needed for repairing the solution per edge update, over a worst-case sequence of edge updates.

Our dynamic maximal hypergraph matching algorithm can be naturally adapted to *distributed networks*. Note that following an edge update, $O(1)$ communication rounds trivially suffice for updating a maximal hypergraph matching. However, the number of messages sent per update via this naive algorithm may be a factor of r^2 greater than the maximum degree in the hypergraph, which could be $\Omega(\binom{n}{r-1})$, where n is the number of vertices and r is the rank. An important objective is to design a dynamic distributed algorithm that achieves, in addition to low round complexity, a low *message complexity*. The inclusion-maximality of our maintained matching enforces our algorithm to work persistently so there is never any “slack”; that is, the algorithm makes sure that every edge that can be added to the matching is added to it, which stands in contrast to “lazy” approximate-maximal matching (or approximate set cover) algorithms, which may wait to accumulate an ε -factor additive slack in size or weight, and only then run an update procedure. However, such a lazy update procedure is inherently non-local, where, following an edge update e , the required changes to the maintained graph structure may involve edges and vertices that are arbitrarily far from e ; indeed, this is the case with the previous algorithms that achieve approximation factor close to f [11, 1]; moreover, the “lazy” feature of these algorithms must rely on a centralized agent that orchestrates the update algorithm with the use of global data structures, and this is, in fact, the key behind the efficiency of these algorithms [11, 1]. Our algorithm, on the other hand, does not employ any global update procedure or global data structures, and as such it is inherently *local* and can be easily distributed, so that the average number of messages sent per update is $O(r^2)$, matching the sequential update time. The number of rounds is clearly upper bounded by the number of messages. Refer to Section 9 in the full version [4] for more details.

Recent related work

Independently and concurrently to our work, Bhattacharya, Henzinger, Nanongkai, and Wu [arXiv’20, SODA’21] obtained an algorithm for dynamic set cover with $O(f^2/\varepsilon^3)$ amortized update time and $O(f \cdot \log^2 n/\varepsilon^3)$ worst case update time and $(1 + \varepsilon) \cdot f$ approximation. While closely related, their results and ours are incomparable. Our algorithm can achieve a better approximation ratio of *exactly* f as opposed to $(1 + \varepsilon)f$ (which is the first dynamic algorithm with this guarantee) but is randomized, while their result is deterministic and can work for weighted set cover (with extra $\log C$ -dependence on the update time where C is the maximum weight of any set). Also, as mentioned, our algorithm is inherently local and can be distributed efficiently, whereas the algorithm of Bhattacharya et al., as the previous aforementioned algorithms, is non-local. In terms of techniques, the two works are disjoint.

1.1 Technical overview

At a high level, all the previous state-of-the-art algorithms [18, 9, 11] (as well as the independent work of [12]) follow a deterministic primal-dual approach by maintaining a fractional packing solution as a dual certificate. The main advantage of this approach over ours, of course, is in being deterministic, and its drawbacks are that (1) the approximation factor is almost f rather than exactly f , (2) it does not give rise to an *integral* matching in the context of hypergraphs, and (3) it is non-local.

Our algorithm generalizes and strengthens the maximal matching algorithm by Solomon [25], which, in turn, builds on and refines the pioneering approach of Baswana et al. [5]. For conciseness, we shall sometimes refer only to [25] in the following (to avoid explaining the differences between [25] and [5]); of course, by that we do not mean to take any credit of [5], on which [25] relies.

Maximal matching algorithm of [5, 25]. Consider a deletion of a matched edge (u, v) from the graph (for $r = 2$), which is the only nontrivial update. Focusing on u , if u has an unmatched neighbor, we need to match u . To avoid a naive scan of the neighbors of u (requiring $O(n)$ time), the key idea is to match u with a randomly sampled (possibly matched) neighbor w . Under the oblivious adversarial model, the expected number of edges incident on u deleted from the graph before the deletion of edge (u, w) is roughly half the “sample space” size of u , i.e., the number of neighbors of u from which we sampled w , which can be viewed as “time token” (or potential value) that is expected to arrive in the future.

To benefit from these tokens, [5, 25] introduces a *leveling scheme* where the *levels* of vertices are exponentially smaller estimates of these potential values: Unmatched vertices have level -1 and matched vertices are assigned the levels of their matched edge, which is roughly the logarithm of the sample space size. This defines a dynamic hierarchical partition of *vertices* into $O(\log n)$ levels.

A key ingredient in the algorithm of [5, 25] is the *sampling rule*: A random mate w is chosen for a vertex u among its neighbors of strictly lower level. Intuitively, if w ’s level is lower than u ’s, then w ’s potential value is much smaller than u ’s, and the newly created matched edge (u, w) provides enough potential value to cover the cost of deleting the old matched edge on w (if any).

Difficulties of going from ordinary graphs to hypergraphs. The main difficulty of extending the prior work in [5, 25] to hypergraphs of rank $r > 2$ has to do with the “vertex-centric” approach taken by these works [5, 25]. For instance, even at the definition level, it is already unclear how to generalize the sampling rule of the algorithm for hypergraphs, even for $r = 3$, since the hyperedges incident on u may consist of endpoints of various levels, some smaller than that of u , some higher. Ideally we would want to sample the matched hyperedge among those where *all* endpoints have lower level than that of u , but it is a-priori unclear how to maintain this hyperedge set efficiently. In particular, to perform the sampling rule efficiently, the strategy of [25] is to dynamically orient each edge towards the lower level endpoint, and a central obstacle is to efficiently cope with edges where both endpoints are at the same level. For hypergraphs, naturally, these obstacles become more intricate considering there are r different endpoints now.

There are also other hurdles that need to be carefully dealt with, such as the following. Say a matched hyperedge $e = (u_1, \dots, u_r)$ gets deleted from the hypergraph. When $r = 2$, any edge incident on u_1 is different than any edge incident on u_2 , hence informally the update algorithm can handle u_1 and u_2 *independently* of each other. When $r > 2$ (even for $r = 3$), different endpoints of e may share (many) hyperedges in common. Therefore, when choosing random matched hyperedges for the newly unmatched endpoints of e , we need to (i) be careful not to create conflicting matched hyperedges, but at the same time (ii) keep the sample spaces of endpoints sufficiently large so that the potential values can cover the runtime of the update procedure; balancing between these two contradictory requirements is a key challenge in our algorithm.

An $O(r^3)$ update time algorithm. We manage to cope with these and other hurdles by instead switching to a “hyperedge-centric” view of the algorithm. Informally speaking, this means that instead of letting vertices derive the potential values and levels, we assign these values to the hyperedges and use those to define the corresponding level for remaining vertices. Under this new view of the algorithm, we can indeed generalize the approach of [25] to obtain an $O(r^4)$ -update time algorithm for rank r hypergraphs. Considering the intricacies in [25], already achieving this $O(r^4)$ time bound turned out to be considerably challenging.

Improving the update time to $O(r^3)$ is based on the following insight. In this hyperedge-centric view, a level- ℓ matched hyperedge e is sampled to the matching from a sample space $S(e)$ of size roughly α^ℓ , where $\alpha = \Theta(r)$. We refer to the hyperedges in $S(e)$ as the *core hyperedges* of e . All the core hyperedges of e are then also assigned a level ℓ . Let us focus on an $e' = (u_1, \dots, u_r) \in S(e)$. Subsequently, u_1 may initiate the creation of a new matched hyperedge at level $\ell' > \ell$, at which stage we randomly sample a new matched hyperedge, denoted by e_1 , among all its incident hyperedges of level lower than ℓ' , so $e' \in S(e_1)$, i.e., hyperedge e' is now a core hyperedge of e_1 as well. Perhaps later u_2 may initiate the creation of a new matched hyperedge e_2 at level $\ell'' > \ell'$, and then hyperedge e' will become a core hyperedge of e_2 , and so on and so forth. Thus any hyperedge may serve as a core hyperedge of up to r matched hyperedges at any point in time.

To shave a factor of r from the runtime, we need to make sure that each hyperedge serves as a core hyperedge of a *single* matched hyperedge. This is achieved by “freezing” (or *temporarily* deleting) all core hyperedges of a newly created matched hyperedge e ; in the sequel (see Section 3.2) we shall refer to these hyperedges as *temporarily deleted hyperedges* (due to e) rather than “core hyperedges”, and they will comprise the set $\mathcal{D}(e)$. Then, whenever the matched hyperedge gets deleted from the matching, we need to “unfreeze” these core hyperedges and update all their *ignored* data structures – our analysis shows that this can be carried out efficiently.

Since this algorithm already requires an entirely new view of the previous approaches for ordinary graphs and several nontrivial ideas, we present it as a standalone result in Section 3.

An $O(r^2)$ update time algorithm. The main step is to improve the update time from $O(r^3)$ to $O(r^2)$. We next provide a couple of technical highlights behind this improvement.

In our algorithm, the potential values of level- ℓ matched hyperedges are in the range $[\alpha^\ell, \alpha^{\ell+1})$, for $\alpha = O(r)$, hence they may differ by a factor of $O(r)$. Consider the moment a level- ℓ matched hyperedge $e = (u_1, \dots, u_r)$ gets deleted by the adversary; the leveling scheme allows us to assume we have an $O(\alpha^{\ell+2})$ “potential time” for handling this hyperedge. To get an update time of $O(r^2)$, we need to handle each of the endpoints $u_i \in e$ within time $O(\alpha^{\ell+1})$ time or instead “contribute” to the potential by creating a new matched hyperedge. If u_i has more than $\alpha^{\ell+1}$ incident hyperedges of level at most ℓ , we can sample a random hyperedge among them to be added to the matching, thereby creating a level- $(\ell+1)$ matched hyperedge; we discuss some issues related to the creation of matched edges later. But if u_i has slightly less than $\alpha^{\ell+1}$ incident edges, this sample space size suffices only for creating a level- ℓ matched edge, but it is crucial that the sample space of a level- ℓ matched edge would consist only of edges where all endpoints have level strictly lower than ℓ . Even checking whether this is the case is too costly, since iterating over all endpoints of all such edges takes time (slightly less than) $O(\alpha^{\ell+2})$, and if we are in the same scenario for each u_i this gives rise to a runtime of $O(\alpha^{\ell+3})$, and thus to an amortized update time of $O(r^3)$. Even if we could check this for free, if most of the edges have endpoints of level ℓ , we need to find those endpoints, and to pass the “ownership” of the edges to those endpoints. To cope with these issues, we maintain a data structure for each hyperedge e that keeps track of all its endpoints of highest level and in $O(1)$ time returns an arbitrary such endpoint or reports that none exists. Of course, now the challenge becomes maintaining these data structures for the edges with $O(r^2)$ update time.

Consider the moment that a level- ℓ matched edge e is created. At this stage, another crucial invariant tells us to “raise” all endpoints of edge e to level ℓ , and then to update the ownership set of each endpoint according to its up-to-date level. One challenging case is when

each endpoint u_i of e now owns slightly less than $\alpha^{\ell+1}$ new edges. This sample space size does not suffice for creating a level- $(\ell + 1)$ matched edge yet it is too costly to update each of the endpoints of these edges about the up-to-date level of u_i ; summing over all endpoints of e , this again gives rise to update time of $O(r^3)$. However, if we are equipped with the aforementioned data structure, we can efficiently focus on the edges where all endpoints have level lower than ℓ , and can thus create a level- ℓ matched edge. This is not enough, however, since we are merely replacing one level- ℓ matched edge by another, and this process could repeat over and over. Our goal would be to replace one level- ℓ matched edge by *at least two others*, so as to provide enough increase in “potential” to cover for this runtime. Since the total number of edges incident on e in this case is slightly less than $\alpha^{\ell+2}$, the intuition is that we should be able to easily achieve here a fan-out of 2, and therefore a valid charging argument. Alas, there is one significant caveat when working with hypergraphs, which we already mentioned above – dependencies. It is possible that the first level- ℓ matched edge that we randomly sampled for u_1 intersects all the edges incident on e , which will result in fan-out 1. To overcome this final obstacle, we take our hyperedge-centric view to the next level by employing a new sampling method different than [5, 25] altogether. In particular, in this case, our sample space is not necessarily restricted to hyperedges incident on a single vertex, but could be an arbitrary hyperedge set as a function of the deleted hyperedge; however, importantly, to achieve a *local* sampling method, we will make sure that the entire sample space is incident to the endpoints of a single edge. This new sampling method (see Procedure `insert-hyperedge` in Section 5 of the full version [4]) entails a few technical complications primarily to ensure that we still get enough “potential” from the adversary, but it ultimately enables us to achieve the desired update time bound of $O(r^2)$.

The role of randomness in our algorithm. Our algorithm relies crucially on randomization and on the oblivious adversary assumption, and the probabilistic analysis employed in this work is highly nontrivial; in particular, the usage of randomization for reducing the update time bound from $O(r^3)$ to $O(r^2)$ relies on several new insights. We note that if the entire update sequence is known in advance and is stored in a data structure that allows for fast access, which is sometimes referred to as the “(dynamic) offline setting” (cf. [16]) – then a straightforward variant of our algorithm works *deterministically* with $O(r^2)$ amortized update time. Specifically, whenever a matched edge is randomly sampled by our algorithm (which is always done uniformly) from a carefully chosen sample space of edges – in the offline setting, instead of randomly sampling the matched edge, one can choose the matched edge *deterministically* to be the one that will be deleted last among all edges in the sample space. It is not difficult to verify that this simple tweak translates our probabilistic $O(r^2)$ amortized update time bound into a deterministic $O(r^2)$ time bound, while avoiding the entire probabilistic analysis. The probabilistic ingredients of the analysis are omitted due to space constraints; they appear in Sections 7 and 8 of the full version [4].

2 Preliminaries and Organization

Hypergraph Notation. For a hypergraph $G = (V, E)$, V denotes the set of vertices and E denotes the set of hyperedges. We use $v \in e$ to mean that v is one of the vertices incident on hyperedge e . Rank r of a hypergraph is the maximum number of vertices incident on any edge, i.e., $r := \max \{|e| \mid e \in E\}$. A matching M in G is a collection of vertex-disjoint hyperedges of M . A matching M is called *maximal* if no other edge of G can be added to M without violating its matching constraint. A vertex cover U in G is a collection of vertices so that every hyperedge in E has at least one endpoint in U . We use the next standard fact.

► **Fact 1.** *Let G be any hypergraph with rank r . Suppose M is a maximal matching in G and U denotes all vertices incident on M . Then U is an r -approximate vertex cover of G .*

Hypergraph Formulation of Set Cover. Consider a family $\mathcal{S} = \{S_1, \dots, S_m\}$ of sets over a universe $[n]$. We can represent \mathcal{S} by a hypergraph G on m vertices corresponding to sets of \mathcal{S} and n hyperedges corresponding to elements of $[n]$: Any element $i \in [n]$ is now a hyperedge between vertices corresponding to sets in \mathcal{S} that contain i . It is easy to see that there is a one-to-one correspondence between set covers of \mathcal{S} and vertex covers of G and that the rank r of G is the same as the maximum frequency parameter f in the set cover instance. With this transformation, by Fact 1, obtaining an f -approximation to this instance of set cover reduces to obtaining a maximal matching of G . This is the direction we take in this paper for designing fully dynamic algorithms for set cover by designing a fully dynamic algorithm for hypergraph maximal matching.

Organization. Due to space constraints, in this extended abstract we focus on the $O(r^3)$ update time algorithm and its analysis, where we provide only part of the analysis. The $O(r^2)$ update time algorithm and its analysis are inherently more involved and are entirely omitted. All the missing details appear in the full version [4].

3 An $O(r^3)$ -Update Time Algorithm

Throughout, we use M to denote the maximal matching of the underlying hypergraph $G = (V, E)$ maintained by the algorithm. We use the following parameters in our algorithm:

$$\alpha := (4 \cdot r), \quad L := \lceil \log_\alpha |N| \rceil \quad (1)$$

where N approximates the dynamic number of edges $|E|$ plus the fixed number of vertices $|V|$ from above, so that $\log_\alpha |N| = \Theta(\log_\alpha (|V| + |E|))$. Every $\Omega(N)$ steps we update the value of N , and as a result rebuild all the data structures; this adds a runtime of $O(|V| + |E|) = O(N)$ every $\Omega(N)$ update steps, hence a negligible overhead to the amortized cost of the algorithm. We may henceforth ignore this technical subtlety and treat N as a fixed value in what follows.

3.1 A Leveling Scheme and Hyperedge Ownerships

We use a leveling scheme for the input hypergraph that partitions hyperedges and vertices. This is done by assigning a *level* $\ell(e)$ to each hyperedge $e \in E$ and a *level* $\ell(v)$ to each vertex $v \in V$.

► **Invariant 2.** *Our leveling scheme satisfies the following properties:*

1. *For any hyperedge $e \in E$, $0 \leq \ell(e) \leq L$ and for any vertex $v \in V$, $-1 \leq \ell(v) \leq L$; moreover, $\ell(v) = -1$ iff v is **unmatched** by M .*
2. *For any **matched** hyperedge $e \in M$ and any incident vertex $v \in e$, $\ell(v) = \ell(e)$.*
3. *For any **unmatched** hyperedge $e \notin M$, $\ell(e) = \max_{v \in e} \ell(v)$.*

Our leveling scheme needs only to specify $\ell(e)$ for each $e \in M$; the rest are fixed deterministically by Invariant 2. Moreover, this invariant ensures that the matching M obtained by the algorithm is maximal as all unmatched vertices are at level -1 while the level of any hyperedge is at least 0 and at the same time equal to the maximum level of any of its incident vertices. Based on the leveling scheme, we assign each hyperedge e to exactly one of its incident vertices $v \in e$ with $\ell(v) = \ell(e)$ to *own* (the ties between multiple vertices at the same level are broken by the algorithm). We use $\mathcal{O}(v)$ to denote the set of edges owned by v . This definition, combined with Invariant 2, implies the following invariant.

► **Invariant 3.** (i) For any vertex $v \in V$, any owned hyperedge $e \in \mathcal{O}(v)$, and any other incident vertex $u \in e$, $\ell(u) \leq \ell(v)$. (ii) For any vertex $v \in V$, any incident hyperedge $e \in v$ has $\ell(e) \geq \ell(v)$.

3.2 Temporarily Deleted Hyperedges

To obtain the desired update time of $O(r^3)$, we would need to allow some hyperedges of the hypergraph to be considered *temporarily deleted* and no longer participate in any of the other data structures; moreover, whenever we no longer consider them deleted, we simply treat them as a hyperedge insertion to our hypergraph and handle them similarly (which will take $O(r)$ time per each hyperedge exactly as in the previous algorithm). The role of these deleted hyperedges becomes apparent only in the probabilistic analysis that is omitted due to space constraints; see Section 7 in the full version [4] for this analysis. However, it is the goal of the algorithm itself (rather than the analysis) to cope efficiently with the required deletions. We shall note that these deletions constitute one of several differences of our algorithm with that of [25]. The next invariant allows us to maintain the maximality of our matching.

► **Invariant 4.** Any *temporarily deleted* hyperedge is incident on some **matched** hyperedge.

To maintain Invariant 4, each matched hyperedge $e \in M$ is *responsible* for a set of deleted edges denoted by $\mathcal{D}(e)$ and stored in a linked-list data structure. This set will be *finalized* at the time e joins the matching M and will remain unchanged throughout the algorithm until e is removed from M ; at that point, we simply bring back all hyperedges in $\mathcal{D}(e)$ to the graph as new hyperedge insertions. Invariant 4 is crucial for the correctness of our algorithm.

We note that besides this data structure $\mathcal{D}(e)$, these deleted hyperedges *do not appear* in any other data structure of the algorithm and do not (necessarily) satisfy any of the invariants in the algorithm – they are simply treated as if they do not belong to the hypergraph. Invariant 4 ensures that even though we are ignoring temporarily deleted hyperedges in the algorithm, the resulting maximal matching on the hypergraph of undeleted hyperedges is still a maximal matching for the entire hypergraph. As such, throughout the rest of the paper, with a slight abuse of notation, whenever we talk about hyperedges of G , we refer to the hyperedges that are not temporarily deleted (unless explicitly stated otherwise).

3.3 Data Structures

We maintain the following information for each vertex $v \in V$ (again, to emphasize, we ignore the temporarily deleted hyperedges in all the following data structures):

- $\ell(v)$: the level of v in the leveling scheme;
- $M(v)$: the hyperedge in M incident on v (if v is unmatched $M(v) = \perp$);
- $\mathcal{O}(v)$: the set of hyperedges e owned by v – we define $o_v := |\mathcal{O}(v)|$;
- $\mathcal{N}(v)$: the set of hyperedges e incident on v ;
- $\mathcal{A}(v, \ell)$ for any integer $\ell \geq \ell(v)$: the set of hyperedges $e \in \mathcal{N}(v)$ that are *not* owned by v and have level $\ell(e) = \ell$ – we define $a_{v, \ell} := |\mathcal{A}(v, \ell)|$.

We also maintain the following information for each hyperedge $e \in E$:

- $\ell(e)$: the level of e in the leveling scheme;
- $O(e)$: the single vertex $v \in e$ that owns e , i.e., $e \in \mathcal{O}(v)$;
- $M(e)$: a Boolean variable to indicate whether or not e is matched.

We also maintain back-and-forth pointers between these different data structures that refer to the same hyperedge or vertex in a straightforward way.

8:10 Fully Dynamic Set Cover via Hypergraph Maximal Matching

Next, we introduce the main procedures used for updating these data structures. In these procedures, if there is room for confusion, we use superscript $^{\text{old}}$ to denote a parameter or data structure $*$ before the update and $^{\text{new}}$ to denote the value of $*$ after the update.

Procedure $\text{set-owner}(e, v)$. Given a hyperedge e and vertex $v \in e$ where $\ell(v) = \max_{u \in e} \ell(u)$, sets owner of e to be v , i.e., $O(e) = v$.

To implement $\text{set-owner}(e, v)$, we first set $O^{\text{new}}(e) = v$ and $\ell^{\text{new}}(e) = \ell(v)$, add e to $\mathcal{O}(v)$, and remove e from $\mathcal{O}(O^{\text{old}}(e))$. If $\ell^{\text{new}}(e) = \ell^{\text{old}}(e)$, there is nothing else to do. Otherwise, for any $w \neq v \in e$, we remove e from $\mathcal{A}(w, \ell^{\text{old}}(e))$ and instead insert e in $\mathcal{A}(w, \ell^{\text{new}}(e))$.

▷ **Claim 5 (straightforward proof).** $\text{set-owner}(e, v)$ takes $O(r)$ time.

Procedure $\text{set-level}(v, \ell)$. Given a vertex $v \in V$ and integer $\ell \in \{-1, 0, \dots, L\}$, updates the level of v to ℓ , i.e., sets $\ell(v) = \ell$.

The implementation of $\text{set-level}(v, \ell)$ is as follows. First, we determine the ownership of all hyperedges $e \in \mathcal{O}^{\text{old}}(v)$ that were previously owned by v . We go over each hyperedge $e \in \mathcal{O}^{\text{old}}(v)$ one by one, find $u := \arg \max_{w \in e} \ell(w)$ (where we use $\ell^{\text{new}}(v) = \ell$ for the computations here), and use the procedure $\text{set-owner}(e, u)$ to update the owner of e to u .

If $\ell^{\text{old}}(v) > \ell$, nothing is left to do as no new hyperedge needs to be owned by v now that level of v has decreased (Invariant 3). If $\ell^{\text{old}}(v) < \ell$, we should make v the new owner of all hyperedges $e \in \mathcal{N}(v)$ with level between $\ell^{\text{old}}(v)$ to $\ell - 1$. This is done by traversing the lists maintained in $\mathcal{A}(v, \ell^{\text{old}}(v)), \dots, \mathcal{A}(v, \ell - 1)$, and running $\text{set-owner}(e, v)$ for each edge e in these lists.

Remark. It will be the responsibility of the procedure calling set-level to make sure that the invariants (and particularly Invariant 2) continue to hold.

▷ **Claim 6.** $\text{set-level}(v, \ell)$ takes $O(r \cdot (o_v^{\text{old}} + o_v^{\text{new}}) + \ell + 1)$ time.

Proof. The algorithm iterates over all hyperedges in $\mathcal{O}^{\text{old}}(v)$ and use set-owner that takes $O(r)$ time by Claim 5 for each one. When $\ell^{\text{old}}(v) > \ell$, this is all that is done by the algorithm and the bound on runtime follows. Otherwise, the algorithm also needs to iterate over the lists $\mathcal{A}(v, \ell^{\text{old}}(v)), \dots, \mathcal{A}(v, \ell - 1)$ one by one which takes $O(\ell)$ time and run set-owner for each of them that takes $O(r \cdot o_v^{\text{new}})$ time in total. This gives the bound on runtime. ◁

3.4 The Update Algorithm

There are multiple cases to handle by the update algorithm depending on whether the updated hyperedge is an insertion or deletion, and whether or not it belongs to the maximal matching M . But the only interesting case is when a hyperedge in M is deleted and that is where we start with.

3.4.1 Case 1: a hyperedge $e \in M$ is deleted

There are two things we should take care of upon deletion of a hyperedge e from M ; bringing back the temporarily deleted hyperedges in $\mathcal{D}(e)$ to maintain Invariant 4, and more importantly, updating the matching M to ensure its maximality. All the interesting part of the algorithm happen in the second step, but before we get to that, let us quickly mention how the first part is done.

Maintaining Invariant 4

Throughout the course of handling a single hyperedge update, we may need to delete multiple hyperedges e_1, e_2, \dots from M , starting from the originally deleted hyperedge by the adversary. Each of these hyperedge $e_i \in M$ that are now removed from M is responsible for temporarily deleted hyperedges $\mathcal{D}(e_i)$.

We will maintain a queue of all hyperedges in $\mathcal{D}(e_1), \mathcal{D}(e_2), \dots$ during the course of this update. At the *very end* of the update, once all other changes are finalized, we will insert the hyperedges in this queue one by one to the hypergraph as if they were inserted by the adversary (using the procedure of Section 3.4.3). This allows us to maintain Invariant 4.

We note that throughout the update we may also temporarily delete some new hyperedges. We'll show that in that case, all these hyperedges are also incident on some matched hyperedge which is responsible for them and thus Invariant 4 holds for these hyperedges as well.

Maintaining maximality of M

Let us now begin the main part of the update algorithm. Suppose $e = (v_1, \dots, v_r)$ is deleted from M . This makes the vertices v_1, \dots, v_r *temporarily* free. We will handle each of these vertices using the procedure **handle-free** which is the key ingredient of the update algorithm (we will simply run **handle-free**(v_1), \dots , **handle-free**(v_r)).

Procedure handle-free(v). Handles a given vertex $v \in V$ which is *unmatched* currently in the algorithm (its matched hyperedge may have been deleted via an update or by the algorithm in this time step, or v may simply be unmatched at this point of the algorithm)^a.

^a To simplify the exposition, we may some time call **handle-free**(v) for a vertex v that is now matched again (while handling other vertices). In that case, this procedure simply aborts.

The execution of Procedure **handle-free**(v) depends on the number of owned hyperedges of v , i.e., o_v (both procedures used within this one are described below): (i) if $o_v < \alpha^{\ell(v)+1}$, we will run **deterministic-settle**(v); and otherwise (ii) if $o_v \geq \alpha^{\ell(v)+1}$, we run **random-settle**(v) instead. We now describe each procedure.

Procedure deterministic-settle(v). Handles a given vertex $v \in V$ with $o_v < \alpha^{\ell(v)+1}$.

In **deterministic-settle**(v), we iterate over all hyperedges $e \in \mathcal{O}(v)$ owned by v and check whether all endpoints of e are now unmatched; if so, we add e to M , and run **set-level**($v, 0$) and **set-owner**(e, v). Moreover, for any vertex $u \neq v \in e$, we further run **set-level**($u, 0$) so all vertices incident on e are now at level 0. If no such hyperedge is found, we run **set-level**($v, -1$).

▷ **Claim 7.** **deterministic-settle**(v) takes $O(r \cdot o_v^{\text{old}}) = O(r \cdot \alpha^{\ell^{\text{old}}(v)+1})$ time and maintains Invariants 2 and 3 for vertex v and hyperedges incident on v .

Proof. Checking if there is any hyperedge that can be added to M takes $O(r \cdot o_v^{\text{old}})$ time. Moreover, running **set-level**($v, 0$) or **set-level**($v, -1$) take $O(r \cdot o_v^{\text{old}})$ time by Claim 6 as $o_v^{\text{new}} \leq o_v^{\text{old}}$ considering level of v has not increased. If the algorithm finds a hyperedge e to add to M , we run **set-level**($u, 0$) for $u \neq v \in e$ as well which takes $O(1)$ time for each u by Claim 6 as $\ell^{\text{old}}(u) = -1$ (u was unmatched and by Invariant 2) and thus $o_u^{\text{old}} = o_u^{\text{new}} = 0$. As there are at most $r - 1$ choices for u , this step takes $O(r) = O(r \cdot \alpha^{\ell^{\text{old}}(v)+1})$ time (we are not being tight here). We also need to run **set-owner**(e, v) in this case which takes another $O(r)$ time by Claim 5. Since **deterministic-settle**(v) is only called when $o_v^{\text{old}} < \alpha^{\ell^{\text{old}}(v)+1}$, the desired time bound follows.

8:12 Fully Dynamic Set Cover via Hypergraph Maximal Matching

As for maintaining Invariant 2, consider the hyperedge e chosen by the algorithm to be added to M . We set the level of all vertices incident on e to be 0 and making v the owner of e , hence satisfying Invariant 2. On the other hand, if we cannot find such a hyperedge e incident on v , we know that all hyperedges incident on v are already at level at least 0 by Invariant 2 (even after removing v) and hence setting $\ell^{\text{new}}(v) = -1$ keeps Invariant 2. Invariant 3 also holds by maintaining Invariant 2 and the fact that we choose correct owners in the algorithm. \triangleleft

Before defining **random-settle**, we need the following definition. For any vertex $v \in V$ and integer $\ell > \ell(v)$, we define:

■ $\tilde{o}_{v,\ell}$: the number of edges v will own *assuming* we increase its level from $\ell(v)$ to ℓ .

The following claim is a straightforward corollary of procedure **set-level**.

▷ **Claim 8.** For any $v \in V$ and $\ell > \ell(v)$, $\tilde{o}_{v,\ell} = \left(\sum_{\ell'=\ell(v)}^{\ell-1} a_{v,\ell'} \right) + o_v$. In particular, $\tilde{o}_{v,\ell}$ can be obtained from $\tilde{o}_{v,\ell-1}$ in $O(1)$ time.

We can now describe **random-settle** (this is the main procedure of our update algorithm).

Procedure random-settle(v). Handles a given vertex $v \in V$ with $o_v \geq \alpha^{\ell(v)+1}$.

In **random-settle**(v), we first compute the level $\ell^{\text{new}}(v)$ as *minimum* $\ell > \ell(v)$ with $\tilde{o}_{v,\ell} < \alpha^{\ell+1}$ and run **set-level**($v, \ell^{\text{new}}(v)$). Then, we sample a hyperedge e uniformly at random from $\mathcal{O}^{\text{new}}(v)$. The next step of the algorithm now depends on this particular hyperedge e .

- **Case (a):** for all $u \in e$, $\tilde{o}_{u,\ell^{\text{new}}(v)} < \alpha^{\ell^{\text{new}}(v)+1}$. In this case, we add the hyperedge e to M and run **set-level**($u, \ell^{\text{new}}(v)$) for all $u \in e$ to maintain Invariant 2. This potentially can make M not a matching since some matched hyperedges e_1, \dots, e_k for $k < r$ incident on e might be in M . We will thus delete these hyperedges from M one by one and *recursively* run the procedure of Section 3.4.1 for each one, treating it *as if* this hyperedge was deleted by the adversary (although we do *not* remove the hyperedge from the hypergraph).
- **Case (b):** there exists $u \in e$, s.t. $\tilde{o}_{u,\ell^{\text{new}}(v)} \geq \alpha^{\ell^{\text{new}}(v)+1}$. We run **deterministic-settle**(v) to handle v and “switch” the focus to u instead. We then call **set-level**($u, \ell^{\text{new}}(v)$). If u is matched in M , say by hyperedge e_u , we remove e_u from M , and then *recursively* run the procedure of Section 3.4.1 for hyperedge e_u – we only note that when processing e_u , we start by running **handle-free**(u) first before all other vertices incident on e_u ; this is only done for making the analysis more clear and is not needed. If u is *not* matched in M , we will simply run **handle-free**(u).

▷ **Claim 9.** The first step of **random-settle**(v) before either case (changing level of v and picking hyperedge e) takes $O(r \cdot \alpha^{\ell^{\text{new}}(v)+1})$ time. Additionally, case (a) takes $O(r^2 \cdot \alpha^{\ell^{\text{new}}(v)+1})$ time and case (b) takes $O(r \cdot o_u^{\text{new}})$ time ignoring the recursive calls. Finally, **random-settle**(v) maintains Invariants 2 and 3.

Proof. Finding $\ell^{\text{new}}(v)$ takes $O(\ell^{\text{new}}(v))$ time by Claim 8 and **set-level**($v, \ell^{\text{new}}(v)$) takes $O(r \cdot o_v^{\text{new}} + \ell^{\text{new}}(v)) = O(r \cdot \alpha^{\ell^{\text{new}}(v)+1})$ time by Claim 6 as level of v is increased (and so is number of its owned edges) and since $o_v^{\text{new}} < \alpha^{\ell^{\text{new}}(v)+1}$ by design. This proves the first part.

The second part for case (a) also follows because for any $u \in e$, **set-level**($u, \ell^{\text{new}}(v)$) takes $O(r \cdot o_u^{\text{new}} + \ell^{\text{new}}(u)) = O(r \cdot \tilde{o}_{u,\ell^{\text{new}}(v)} + \ell^{\text{new}}(v))$ by Claim 6 as level of u is increased. This is $O(r \cdot \alpha^{\ell^{\text{new}}(v)+1})$ by the condition $\tilde{o}_{u,\ell^{\text{new}}(v)} < \alpha^{\ell^{\text{new}}(v)+1}$ in case (a). Multiplying this with at most r vertices $u \in e$ gives the desired bound.

The second part for case (b) holds because v has $o_v \leq \alpha^{\ell^{\text{new}}(v)+1}$ by the definition of $\ell^{\text{new}}(v)$ and thus **deterministic-settle**(v) takes $O(r \cdot \alpha^{\ell^{\text{new}}(v)+1})$ time by Claim 7. Moreover, running **set-level**($u, \ell^{\text{new}}(v)$) takes $O(r \cdot o_u^{\text{new}})$ by Claim 6 which is at least $O(r \cdot \alpha^{\ell^{\text{new}}(v)+1})$ by the lower bound on $\tilde{o}_{u, \ell^{\text{new}}(v)}$ in this case.

Finally, Invariants 2 and 3 also hold as explained in the description of the procedure. \triangleleft

The following observation plays a key rule in the recursive analysis of our algorithm.

► **Observation 10.** *In **random-settle**(v): (i) the hyperedge e is sampled uniformly at random from at least $\alpha^{\ell^{\text{new}}(v)}$ edges. Moreover, (ii) any hyperedge e_1, \dots, e_k or e_u deleted from M during this procedure is at level at most $\ell^{\text{new}}(v) - 1$.*

Part (i) of the observation follows from the definition of $\ell^{\text{new}}(v)$. For part (ii), note that any deleted edge e' in the process is incident on e with $\ell^{\text{old}}(e) = \ell^{\text{old}}(v) < \ell^{\text{new}}(v)$. Let u be any vertex incident on both e' and e . Firstly, $\ell(u) \leq \ell^{\text{old}}(v)$ as e was owned by v and not u , and secondly $\ell(e') = \ell(u)$ as e' is a matched hyperedge (see Invariant 2); thus $\ell(e') \leq \ell(e) < \ell^{\text{new}}(v)$ as well.

Temporarily deleting new hyperedges

An astute reader may have noticed that we have not yet temporarily deleted any hyperedge in the update algorithm. The only place that this will be done is in case (a) of **random-settle**. In this case, when we decide to insert e to M , we will temporarily delete all other hyperedges in $\mathcal{O}^{\text{new}}(v)$ from which e was sampled from, and make e responsible for them by adding them to $\mathcal{D}(e)$. The deletions of these hyperedges is done by the procedure of Section 3.4.2 (as they do not belong to M). As the cost of each such hyperedge deletion is $O(r)$ and their total number is $O(\alpha^{\ell^{\text{new}}(v)+1})$, this does not change the asymptotic bounds of Claim 9. Finally, since these edges are incident on e which now belongs to M , Invariant 4 will be maintained by this process. The following observation is now immediate.

► **Observation 11.** *Any hyperedge $e \in M$ is responsible for $O(\alpha^{\ell(e)+1})$ hyperedges in $\mathcal{D}(e)$.*

3.4.2 Case 2: a hyperedge $e \notin M$ is deleted

We only need to remove e from the corresponding data structures which can be done in $O(r)$ time for the (at most) r endpoints of e .

3.4.3 Case 3: a hyperedge e is inserted

We need to find $v = \arg \max_{u \in e} \ell(u)$ and run **set-owner**(e, v) which takes $O(r)$ time. Moreover, if all vertices incident on e are unmatched, we should additionally add e to M and run **set-level**($u, 0$) for all $u \in e$ which takes $O(r)$ time per vertex by Claim 6.

This concludes the description of our algorithm for update time $O(r^3)$.

4 Part of the Analysis of the $O(r^3)$ -Update Time Algorithm

A key definition in analysis is the notion of an *epoch* borrowed from the prior work in [5, 25].

► **Definition 12 (Epoch).** *For any time t and any hyperedge e in the matching M at time t , **epoch** of e and t , denoted by $\text{epoch}(e, t)$, is the pair $(e, \{t'\})$ where $\{t'\}$ denotes the maximal continuous time period containing t during which e was always present in M (not even deleted temporarily during one step and inserted back at the same time step).*

We further define level of $\text{epoch}(e, t)$ as the level $\ell(e)$ of e during the time period of the epoch; we note that our update algorithm never changes level of a hyperedge in M without removing it from M first and thus level of an epoch is well-defined.

The update algorithm in Section 3.4 takes $O(r)$ time deterministically for any update step that does *not* change the matching M . However, an update at a time step t that changes M may force the algorithm a large computation time and hence we use amortization to charge the cost of processing such an update at time t to the epochs that are created and removed at time t . In particular, for any time step t , define:

- $\text{epochs}_{\text{create}}(t)$: the set of all $\text{epoch}(e, t)$ that are *created* at time t ; respectively, $\text{epochs}_{\text{term}}(t)$ is defined for epochs *terminated* at time t ;
- $C_{\text{create}}(\text{epoch}(e, t))$: the computation cost at time t for *creation* of $\text{epoch}(e, t)$; respectively, $C_{\text{term}}(\text{epoch}(e, t))$ is defined for the cost of *termination* of $\text{epoch}(e, t)$.

Then, the total cost of update at time t is:

$$C(t) = O(r) + \sum_{\substack{\text{epoch}(e, t) \in \\ \text{epochs}_{\text{create}}(t)}} C_{\text{create}}(\text{epoch}(e, t)) + \sum_{\substack{\text{epoch}(e, t) \in \\ \text{epochs}_{\text{term}}(t)}} C_{\text{term}}(\text{epoch}(e, t)). \quad (2)$$

► **Lemma 13.** *The total computation cost $C(t)$ of an update at time t in Equation (2) can be charged to the epochs in the RHS so that any level- ℓ epoch is charged with $O(\alpha^{\ell+3})$ units of cost.*

Proof. We prove this lemma in a sequence of claims. In the following, whenever we say “some computation cost can be charged to $C_{\text{create}}(\text{epoch}(e, t))$ or $C_{\text{term}}(\text{epoch}(e, t))$ ” we mean that the cost of computation is $O(\alpha^{\ell+3})$ and there is a level- ℓ epoch $\text{epoch}(e, t)$ at time t that is either created or terminated, respectively, and that this cost is charged to $C_{\text{create}}(\text{epoch}(e, t))$ or $C_{\text{term}}(\text{epoch}(e, t))$. We emphasize that each epoch during this process is only charged a *constant* number of times. This then immediately satisfies the bounds in Lemma 13.

▷ **Claim 14.** Cost of a hyperedge-insertion update e at time t can be charged to $C_{\text{create}}(\text{epoch}(e, t))$.

Proof. As in Section 3.4.3, this update takes $O(r)$ time and creates a level-0 $\text{epoch}(e, t)$. ◁

From now on, we consider the main case of a hyperedge-deletion update e from M at time t using the procedure in Section 3.4.1. In this case, we remove e from M which results in $\text{epoch}(e, t)$ to be terminated (and hence $\text{epoch}(e, t) \in \text{epochs}_{\text{term}}(t)$ in RHS of $C(t)$ in Equation (2)). This step is then followed by running **handle-free**(v) for all $v \in e$. Recall that **handle-free**(v) can be handled either by **deterministic-settle**(v) or **random-settle**(v) (depending on value of o_v).

▷ **Claim 15.** Let e be a hyperedge in M deleted at time t (either by the adversary or the algorithm in a recursive call). Cost of **handle-free**(v) for all $v \in e$ that are handled by **deterministic-settle**(v) can be charged to $C_{\text{term}}(\text{epoch}(e, t))$.

Proof. By Claim 7, for each $v \in e$ that is handled by **deterministic-settle**(v), the runtime of **handle-free**(v) is $O(r \cdot \alpha^{\ell(e)+1})$. As there are at most r such vertices, their total cost is $O(r^2 \cdot \alpha^{\ell(e)+1}) = O(\alpha^{\ell(e)+3})$ by the choice of α in Equation (1). Hence the total cost of all these vertices that are charged to $C_{\text{term}}(\text{epoch}(e, t))$ is $O(\alpha^{\ell(e)+3})$. ◁

We now switch to analyzing **random-settle**(v) for a fixed $v \in e$. The easier part, when we can process **random-settle**(v) by case (a), is handled via the next claim.

▷ **Claim 16.** Cost of case (a) of **random-settle**(v) at time t ignoring the recursive calls can be charged to $C_{create}(\text{epoch}(e_v, t))$, where e_v is the hyperedge inserted to M in this case.

Proof. In case (a) of **random-settle**(v), we create a new edge e_v in M at level $\ell(e_v) = \ell^{\text{new}}(v)$. By Claim 9, ignoring the recursive calls, this case takes $O(r^2 \cdot \alpha^{\ell(e_v)+1}) = O(\alpha^{\ell(e_v)+3})$ time. We can thus charge the cost of **random-settle**(v) to $C_{create}(\text{epoch}(e_v, t))$. ◁

We now analyze case (b) of **random-settle**(v). In this case, instead of handling v , we in fact recursively handle the vertex u (with $\tilde{o}_{u, \ell^{\text{new}}(v)} \geq \alpha^{\ell^{\text{new}}(v)+1}$) using **handle-free**(u) and only after that will come back to take care of v if needed. Moreover, since we first set level of u to $\ell^{\text{new}}(v)$, we will have $o_u^{\text{new}} \geq \alpha^{\ell^{\text{new}}(u)+1}$. This means **handle-free**(u) will be handled using **random-settle**(u) recursively. It is again possible that **random-settle**(u) hits case (b) and so on and so forth. However, note that when going from v to u , we obtain that $\ell^{\text{new}}(v) < \ell^{\text{new}}(u)$ by Observation 10 (here $\ell^{\text{new}}(u)$ refers to $\ell(u) = \ell^{\text{new}}(v)$ at the beginning of **random-settle**(u)) and hence whenever case (b) happens, the vertex we move on to will have a strictly larger level. As such, this chain of recursive calls to **random-settle** will terminate eventually in some case (a) of **random-settle**.

We'll denote the vertices in the chain of calls to **random-settle** in case (b) for v by $(v =)w_0, w_1, w_2, \dots, w_k$ for some $k \leq L$ (number of levels); in particular, w_1 is the vertex u in case (b) of **random-settle**(v), w_2 is the vertex u in case (b) of **random-settle**(w_1), etc.

The computation cost of **random-settle**(v) in case (b) then involves two parts: (i) pre-processing before calling each **random-settle**(w_i) (which is bounded in Claim 9), (ii) calling **random-settle**(w_i) itself, and (iii) removing hyperedges e_{w_i} from M (for each one that exist) and recursively handling them using the procedure of Section 3.4.1.

Among these costs, the cost of handling e_{w_i} for $i \in [k]$ is handled separately by recursion (charged either to the termination of $\text{epoch}(e_{w_i}, t)$ or creation of new epochs). We thus need to handle the costs in parts (i) and (ii) in the following claim, whose proof is omitted.

▷ **Claim 17.** Cost of case (b) of **random-settle**(v) at time t with chain of vertices w_0, w_1, \dots, w_k ignoring the recursive calls (i.e., part (iii) of costs above) can be charged to $C_{create}(\text{epoch}(e^*, t))$, where e^* is the hyperedge inserted to M in case (a) of **random-settle**(w_k). (Note that by definition, **random-settle**(w_k) finishes in case (a).)

Finally, we also have to handle the cost associated with maintaining Invariant 4, namely, “bringing back” temporarily deleted hyperedges at the very end of the update step (the cost of temporarily deleting new hyperedges is accounted for in Claim 9 already).

▷ **Claim 18.** Cost of inserting back the temporarily deleted hyperedges in $\mathcal{D}(e)$ for any hyperedge $e \in M$ can be charged to $C_{term}(\text{epoch}(e, t))$.

Proof. Any hyperedge $e \in M$ is responsible for $O(\alpha^{\ell(e)+1})$ hyperedges in $\mathcal{D}(e)$ by Observation 11. As bringing back these hyperedges requires $O(r \cdot \alpha^{\ell(e)+1})$ time in total, this charge can be charged to the termination of $\text{epoch}(e, t)$. ◁

We thus showed that any cost in $C(t)$ can be charged by a factor of $O(\alpha^{\ell+3})$ to some level- ℓ epoch in RHS of Equation (2) without charging any epoch more than a constant number of times, thus finalizing the proof of Lemma 13. ◀

In Lemma 13, we charge at most $O(\alpha^{\ell+3})$ to the creation and/or termination of a level- ℓ epoch. It can be shown that, without loss of generality, we may assume that every epoch created will also be terminated, hence we can re-distribute the charges to creation of each level- ℓ epoch to the termination of the same epoch. We now have an equivalent charging

scheme in which *only termination* of each level- ℓ epoch is charged with $O(\alpha^{\ell+3})$ computation time (and not the creations). Recall that an epoch is terminated when the corresponding hyperedge e is deleted from the maximal matching M . There are two types of hyperedge deletions from M in the algorithm: those that are the result of the adversary deleting a hyperedge from the graph, and those that are the result of the update algorithm to remove a matched hyperedge in favor of another (so the original hyperedge is still part of the hypergraph). Based on this, we differentiate between epochs as follows:

- **Natural epoch:** Ending with the adversary deleting the hyperedge e from G .
 - **Induced epoch:** Ending with the update algorithm removing the hyperedge e from M .
- With the following lemma, we are going to re-distribute the charge assigned to all induced epochs between the natural epochs. This then will allow us to focus solely on natural epochs.

► **Lemma 19** (proof omitted). *The total cost charged to all induced epochs can be charged to natural epochs so that any level- ℓ natural epoch is charged with the costs of at most $r - 1$ induced epochs at lower levels.*

Next, we move from natural and induced epochs to the corresponding notions in *levels*: A level ℓ is called an **induced level** (resp., **natural level**) if the number of induced level- ℓ epochs is greater than (resp., at most) the number of natural level- ℓ epochs. We will charge the computation costs incurred by any induced level to the computation costs at higher levels, so that in the end, the entire cost of the algorithm will be charged to natural levels. Specifically, in any induced level ℓ , we define a one-to-one mapping from the natural to the induced epochs. For each induced epoch, at most one natural epoch (at the same level) is mapped to it; any natural epoch that is mapped to an induced epoch is called *semi-natural*. For any induced level ℓ , all natural ℓ -level epochs are semi-natural by definition. For any natural level, all natural epochs terminated at that level remain as before; these epochs are called *fully-natural*. By Lemma 19, for any epoch, at most $r - 1$ induced epochs at lower levels are charged to it. We define the **recursive cost** of an epoch as the sum of its actual cost and the recursive costs of the at most $r - 1$ induced epochs charged to it as well as the (at most) $r - 1$ semi-natural epochs mapped to them; the recursive cost of a level-0 epoch is defined as its actual cost. We are now ready to state the following lemma (proof omitted).

► **Lemma 20.** *For any $\ell \geq 0$, the recursive cost of any level- ℓ epoch is bounded by $O(\alpha^{\ell+3})$.*

The sum of recursive costs over all fully-natural epochs is equal to the sum of actual costs over all epochs (fully-natural, semi-natural and induced) that have been *terminated* throughout the update sequence, which also bounds the total runtime of the algorithm.

For the omitted proofs in the above statements, we refer the reader to Section 4 in the full version [4]. The final step of the analysis is to use the randomization in the algorithm (and obliviousness of the adversary) to prove a probabilistic upper bound on the amortized cost of the algorithm; this part is handled in detail in Section 7 of the full version [4].

This allows us to conclude the following theorem.

► **Theorem 21.** *For any integer $r > 1$, starting from an empty rank- r hypergraph on a fixed set of vertices, a maximal matching can be maintained over any sequence of t hyperedge insertions and deletions in $O(t \cdot r^3)$ time in expectation and with high probability.*

References

- 1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 114–125, 2019.

- 2 Shiri Antaki, Quanquan C. Liu, and Shay Solomon. Dynamic distributed MIS with improved bounds. *CoRR*, abs/2010.16177, 2020. [arXiv:2010.16177](#).
- 3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 815–826, 2018.
- 4 Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. *arXiv preprint arXiv:2105.06889*, 2021.
- 5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, October 23-25, 2011*, pages 383–392, 2011 (see also *SICOMP'15* version, and subsequent erratum).
- 6 S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 785–804, 2015.
- 7 S. Bhattacharya, M. Henzinger, and D. Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, 2016.
- 8 S. Bhattacharya, M. Henzinger, and D. Nanongkai. Fully dynamic maximum matching and vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, January 16-19, 2017*, pages 470–489, 2017.
- 9 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in $O(1)$ amortized update time. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017*, pages 86–98, 2017.
- 10 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, January 4-6, 2015*, pages 785–804, 2015.
- 11 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–423. IEEE, 2019.
- 12 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. An improved algorithm for dynamic set cover. *CoRR*, abs/2002.11171, 2020.
- 13 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2 + \epsilon)$ -approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *SODA*, 2019.
- 14 Matthias Bonne and Keren Censor-Hillel. Distributed detection of cliques in dynamic networks. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *Proc. 46th ICALP*, volume 132 of *LIPIcs*, pages 132:1–132:15, 2019.
- 15 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016.
- 16 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proc. 45th ICALP*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.
- 18 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT*

- Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550, 2017.
- 19 Haim Kaplan and Shay Solomon. Dynamic representations of sparse distributed networks: A locality-sensitive approach. In Christian Scheideler and Jeremy T. Fineman, editors, *Proc. of 30th SPAA 2018*, pages 33–42, 2018.
 - 20 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.
 - 21 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.
 - 22 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 220–239, 2016.
 - 23 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.
 - 24 D. Peleg and S. Solomon. Dynamic $(1 + \epsilon)$ -approximate matchings: A density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016.
 - 25 S. Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, New Brunswick, NJ, USA, October 9-11, 2016*, pages 325–334, 2016.
 - 26 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

The Randomized Competitive Ratio of Weighted k -Server Is at Least Exponential

Nikhil Ayyadevara ✉

Indian Institute of Technology, New Delhi, India

Ashish Chiplunkar ✉

Indian Institute of Technology, New Delhi, India

Abstract

The weighted k -server problem is a natural generalization of the k -server problem in which the cost incurred in moving a server is the distance traveled times the weight of the server. Even after almost three decades since the seminal work of Fiat and Ricklin (1994), the competitive ratio of this problem remains poorly understood even on the simplest class of metric spaces – the uniform metric spaces. In particular, in the case of randomized algorithms against the oblivious adversary, neither a better upper bound than the doubly exponential deterministic upper bound, nor a better lower bound than the logarithmic lower bound of unweighted k -server, is known. In this paper, we make significant progress towards understanding the randomized competitive ratio of weighted k -server on uniform metrics. We cut down the triply exponential gap between the upper and lower bound to a singly exponential gap by proving that the competitive ratio is at least exponential in k , substantially improving on the previously known lower bound of about $\ln k$.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases weighted k -server, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.9

Related Version Full Version: <https://arxiv.org/abs/2102.11119>

Funding Ashish Chiplunkar: partially supported by the Pankaj Gupta New Faculty Fellowship.

1 Introduction

The k -server problem of Manasse, McGeoch, and Sleator [12] is one of the cleanest, simple-looking, and yet profound problems in online computation, and has been actively studied for over three decades. The k -server problem concerns deciding movements of k mobile servers on an underlying metric space to serve a sequence of online requests. Each request is issued at some point of the metric space, and to serve such a request, a server must move to the requested point (unless a server is already present there). The cost incurred in the movement of a server is equal to the distance through which the server moves, and the goal is to minimize the total cost.

Since an online algorithm is required to take its decisions only based on the past inputs, it cannot output the optimal solution, in general. An online algorithm for a minimization problem is said to be c -competitive if, on any instance, it produces a solution whose (expected) cost is at most c times the cost of the optimum solution. The competitive ratio of an algorithm is the minimum (technically, the infimum of all) c such that the algorithm is c -competitive. The deterministic (resp. randomized) competitive ratio of an online minimization problem is the minimum (technically, the infimum of all) c for which a c -competitive deterministic (resp. randomized) algorithm exists. Note that, unless otherwise specified, we assume that in case of randomized algorithms, the adversarial input is *oblivious*, that is, constructed with the knowledge of the algorithm but not the random choices the algorithm makes.



© Nikhil Ayyadevara and Ashish Chiplunkar;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 9; pp. 9:1–9:11

Leibniz International Proceedings in Informatics



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

In their seminal work, Manasse, McGeoch, and Sleator [12] proved that the deterministic competitive ratio of the k -server problem is at least k on every metric with more than k points. They conjectured that the deterministic competitive ratio is, in fact, equal to k on any metric. This conjecture is popularly called the deterministic k -server conjecture and it remains unresolved to date. The deterministic algorithm with the best known competitive ratio of $2k - 1$ is due to Koutsoupias and Papadimitriou [10]. Surprisingly, no better algorithm is known even using randomization. The randomized k -server conjecture states that a randomized algorithm with competitive ratio $O(\log k)$ exists on all metrics, and this remains unresolved after some recent progress [5, 11]. The k -server problem on uniform metric spaces is particularly interesting because it is equivalent to the paging problem. In this case, several deterministic algorithms including Least-Recently-Used (LRU) and First-In-First-Out (FIFO) are known to be k -competitive. The randomized competitive ratio is known to be exactly $H(k) = \sum_{i=1}^k 1/i \approx \ln k$, where the lower bound is due to Fiat et al. [8] and the upper bound is due to [13, 1].

The weighted k -server problem is a natural generalization of the k -server problem where the objective is to minimize the weighted sum of the movements of servers. Specifically, the k servers have weights $\beta_1 \leq \dots \leq \beta_k$, and the cost of moving the i 'th server is β_i times the distance through which it moves. It is easy to see that a c -competitive k -server algorithm has competitive ratio at most $c\beta_k/\beta_1$ for the weighted k -server problem, and therefore, the challenge is to design an algorithm with competitive ratio independent of the servers' weights. Surprisingly, this innocuous-looking introduction of weights into the k -server problem makes it incredibly difficult, and a competitive algorithm is known only for $k \leq 2$ [14] (of which, the $k = 1$ case is trivial).

1.1 Weighted k -Server on Uniform Metrics

Owing to the difficulty of the weighted k -server problem on general metrics, the problem becomes particularly interesting on uniform metrics. The weighted k -server problem on uniform metric spaces models the paging problem where the cost of page replacement is determined by the location where the replacement takes place. Note that this problem is different from weighted caching [15], where the cost of page replacement is determined by the pages that get swapped in and out.

The seminal paper of Fiat and Ricklin [9] gave a deterministic algorithm for weighted k -server on uniform metrics whose competitive ratio is doubly exponential in k : about $3^{4^k/3}$ specifically, but can be improved to $2^{2^{k+2}} = 16^{2^k}$ due to the result of Bansal et al. [3] for a more general problem. The fact that the deterministic competitive ratio is indeed doubly exponential in k was established only recently by Bansal et al. [2], who proved a lower bound of $2^{2^{k-4}}$, improving the previously known lower bound of $(k+1)!/2$ due to Fiat and Ricklin [9].

The only known algorithm for the weighted k -server problem on uniform metrics which makes non-trivial use of randomness is by Chiplunkar and Vishwanathan [6]. This algorithm also achieves a doubly exponential competitive ratio of about c^{2^k} for $c \approx 1.59792$. It is, in fact, a *randomized memoryless algorithm* generalizing the algorithm by Chrobak and Sgall [7] for $k = 2$, and it achieves the competitive ratio against a stronger form of adversary called *adaptive online adversary*¹. Chiplunkar and Vishwanathan also proved

¹ An adaptive online adversary can see the movements of the algorithm's servers even though the algorithm is randomized. However, the adversary must also serve its requests in an online manner. The algorithm's

that no randomized memoryless algorithm can achieve a better competitive ratio against adaptive online adversaries. However, even when an algorithm is allowed to use both memory and randomness, and the adversary is oblivious, no better upper bound is known. More embarrassingly, for randomized algorithms, no better lower bound than the logarithmic lower bound of (unweighted) k -server on uniform metrics is known, thus, leaving a triply exponential gap between the upper and lower bounds.

In this paper, we cut down the triply exponential gap between the best known bounds on the randomized competitive ratio of weighted k -server on uniform metrics by a doubly exponential improvement in the lower bound. We prove,

► **Theorem 1.** *The competitive ratio of any randomized algorithm for weighted k -server on uniform metrics is at least exponential in k , even when the algorithm is allowed to use memory and the adversary is oblivious.*

Due to our result, we now have only a singly exponential gap between the best known upper and lower bounds on the randomized competitive ratio of weighted k -server on uniform metrics.

1.2 Comparison with the Deterministic Lower Bound

Our proof of the randomized lower bound for weighted k -server is inspired by the proof of the deterministic lower bound by Bansal et al. [2]. Both proofs give adversaries which run recursively defined strategies relying crucially on a certain set-system \mathcal{Q} . However, our proof differs in the following aspects.

1. The adversary in the deterministic lower bound proof is able to carefully pick from \mathcal{Q} a set of points that does not contain points covered by the algorithm's heavier servers, and run its strategy on that set. In contrast, our adversary is oblivious and is unable to see the positions of the algorithm's servers. Therefore, it merely picks a random set from \mathcal{Q} and hopes that none of the points in that set is covered by the algorithm's heavier servers.
2. The strategy of Bansal et al. to defeat deterministic algorithms ensures that whenever an adversary's server other than the heaviest moves, it is accompanied by an eventual movement of a heavier server of the algorithm. Therefore, assuming that the weights of the servers are well-separated, their task reduces to proving that the heaviest server of the algorithm moves a large number of times as compared to the heaviest server of the adversary. On the other hand, we are unable to charge the movement of an adversary's server to the movement of an algorithm's heavier server. Consequently, we need to carefully track the contributions of all k servers towards the algorithm's and the adversary's costs.

2 Preliminaries

Let the weights of the k servers be $1, \beta, \beta^2, \dots, \beta^{k-1}$ for some large integer β which we will fix later. Define the sequence n_0, n_1, \dots inductively as follows. $n_0 = 1$, and for $\ell > 0$,

$$n_\ell = \left(\left\lceil \frac{n_{\ell-1}}{2} \right\rceil + 1 \right) \cdot \left(\left\lfloor \frac{n_{\ell-1}}{2} \right\rfloor + 1 \right).$$

cost is compared with the cost of the adversary's online solution to determine the competitive ratio.

Observe that n_k grows doubly exponentially with respect to k . Since $n_\ell \geq n_{\ell-1}^2/4$, it is easy to prove using induction that $n_\ell \geq 4 \cdot (64^{1/32})^{2^\ell}$ for all $\ell \geq 5$. Let H denote the harmonic function, that is, $H(n) = \sum_{i=1}^n 1/i$. It is known that $H(n) \geq \ln n$. We will establish Theorem 1 by proving the following bound.

► **Theorem 2.** *The randomized competitive ratio of weighted k -server on uniform metric spaces is at least $H(n_{k-1}) = \Omega(2^k)$.*

We use the following version of Yao's principle to prove the above bound.

► **Proposition 3 (Yao's principle).** *Suppose there exists a probability distribution \mathcal{D} on the instances of an online minimization problem such that for every deterministic online algorithm A , we have,*

$$\mathbb{E}_{I \sim \mathcal{D}}[A(I)] > \alpha \cdot \mathbb{E}_{I \sim \mathcal{D}}[OPT(I)],$$

where $A(I)$ is the cost of the algorithm's solution and $OPT(I)$ is the cost of an optimal solution to instance I . Then the problem does not have an α -competitive randomized online algorithm.

Thus, in order to prove Theorem 2, our task is exhibit a distribution on instances of weighted k -server on a uniform metric space such that the expected cost of any deterministic online algorithm is greater than $H(n_{k-1})$ times the expectation of the optimum cost. To construct our distribution on instances, we use a combinatorial result with a constructive proof given by Bansal et al. [2]. We reproduce its proof in Appendix A for completeness. The result is as follows.

► **Lemma 4.** *Let $\ell \in \mathbb{N}$ and let P be a set of n_ℓ points. There exists a set-system $\mathcal{Q}_\ell \subseteq 2^P$ satisfying the following properties.*

1. \mathcal{Q}_ℓ contains $\lceil n_{\ell-1}/2 \rceil + 1$ sets, each of size $n_{\ell-1}$.
2. For every $p \in P$, there exists a set in \mathcal{Q}_ℓ not containing p .
3. For every $p \in P$, there exists a $q \in P$ such that every set in \mathcal{Q}_ℓ contains at least one of p and q .

3 Adversarial Strategy and Analysis

Consider the uniform metric space on a set S of $n_{k-1} + 1$ points. Our adversarial input distribution is generated by the procedure **adversary** which uses a recursive procedure **strategy**, an oblivious version of its counterpart in Bansal et al. [2]. These procedures are defined as follows.

■ **Procedure 1** **adversary.**

repeat *infinitely many times*

- ┌ Pick a point p uniformly at random from S (with replacement);
 - └ Call **strategy**($k - 1, S \setminus \{p\}$);
-

■ **Procedure 2** $\text{strategy}(\ell, P)$ (Promise: $|P| = n_\ell$).

```

if  $\ell = 0$  (and therefore,  $|P| = n_0 = 1$ ) then
  Request the unique point in  $P$ ;
else
  Construct the set-system  $\mathcal{Q}_\ell \subseteq 2^P$  using Lemma 4;
  repeat  $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1)$  times
    Pick a set  $P'$  uniformly at random from  $\mathcal{Q}_\ell$  (with replacement);
    Call  $\text{strategy}(\ell - 1, P')$ ;

```

Procedure **strategy** gets as input a non-negative number ℓ and a set P of n_ℓ points. In the base case where $\ell = 0$, the procedure issues a request to the unique point in P . In the inductive case where $\ell > 0$, the procedure constructs the set-system \mathcal{Q}_ℓ with properties stated in Lemma 4 on the set P . Then it repeatedly gives recursive calls, passing $\ell - 1$ in place of ℓ , on sets chosen uniformly at random from \mathcal{Q}_ℓ . Recall that these sets have size $n_{\ell-1}$, as required. Procedure **adversary** takes a uniform metric space on $n_{k-1} + 1$ points. It repeatedly picks a point p uniformly at random and calls the procedure **strategy** on the set of points other than p .

For analysis, fix an arbitrary deterministic online algorithm and the initial positions of its servers. We first consider requests given by one execution of procedure $\text{strategy}(\ell, P)$, and bound the number of movements of the algorithm's servers to serve those requests.

► **Lemma 5.** *For every $\ell \in \{0, \dots, k-1\}$ the following holds. Let ρ_0 be an arbitrary sequence of requests and L be the set of positions of the algorithm's heaviest $k - \ell$ servers after serving ρ_0 . Let P be an arbitrary set of n_ℓ points disjoint from L . Suppose ρ_0 is followed by a random sequence ρ of requests given by a $\text{strategy}(\ell, P)$ call. For $i = 1, \dots, k$, let the random variable X_i denote the number of movements of the algorithm's i 'th lightest server while the algorithm serves ρ . Then we have,*

$$\sum_{i=1}^k \beta^{\min(i-1, \ell)} \cdot \mathbb{E}[X_i] \geq (\beta - 1)^\ell.$$

We defer the proof of this lemma to Appendix B. On a high level, the proof goes as follows. If the algorithm moves one of its heaviest $k - \ell$ servers while it serves ρ , then it pays a lot already. If not, it must serve ρ using its lightest ℓ servers only. In this case, each recursive call given by the $\text{strategy}(\ell, P)$ call is, with sufficient probability, on a set P' not containing the location of the algorithm's ℓ 'th lightest server. This enables us to use induction hypothesis to bound the algorithm's cost in each recursive call.

Intuitively, Lemma 5 gives a lower bound of $(\beta - 1)^\ell$ on the expected cost incurred by the algorithm in serving requests given by a $\text{strategy}(\ell, P)$ call, but with the following caveat: movements of the heaviest $k - \ell - 1$ servers are charged at a discounted rate of β^ℓ . However, when ℓ is instantiated to $k - 1$ in particular, no discount remains applicable. Thus, $(\beta - 1)^{k-1}$ becomes a lower bound on the expected actual cost of the algorithm in serving requests given by a $\text{strategy}(k - 1, P)$ call. With this observation, we immediately get the following bound on the expected cost of the algorithm in serving requests given by each **strategy** call made by the procedure **adversary**.

► **Corollary 6** (to Lemma 5). *The expected cost of the algorithm in serving requests given by each **strategy** call made by **adversary** is at least $(\beta - 1)^{k-1} / (n_{k-1} + 1)$.*

Proof. Consider any $\text{strategy}(k-1, S \setminus \{p\})$ call, where p is a uniformly random point in S . Let r be the location of the algorithm's heaviest server at the time the call is made. Then $\Pr[r \notin S \setminus \{p\}] = \Pr[p = r] = 1/|S| = 1/(n_{k-1} + 1)$. Lemma 5 implies that conditioned on $r \notin S \setminus \{p\}$, the expected cost of the algorithm in serving requests given by the $\text{strategy}(k-1, S \setminus \{p\})$ call is at least $(\beta - 1)^{k-1}$. Thus, the claim follows. \blacktriangleleft

Let us now turn our attention towards the adversary's cost. We will show how the adversary, having the ability to see the future requests, can ensure that whenever $\text{strategy}(\ell, P)$ is called, it has at least one server other than its ℓ lightest servers occupying a point in P already. On the contrary, recall that in Lemma 5, we relied on the algorithm not having any of its servers except the ℓ lightest ones occupying points in P at the time $\text{strategy}(\ell, P)$ is called. Intuitively, the adversary is able to obtain advantage over the algorithm by having one server other than the ℓ lightest ones in P whereas the algorithm has none.

► **Lemma 7.** *Define the sequence c_0, c_1, \dots inductively as follows: $c_0 = 0$, and for $\ell > 0$,*

$$c_\ell = \beta^{\ell-1} + \beta \cdot (\lceil n_{\ell-1}/2 \rceil + 1) \cdot c_{\ell-1}.$$

Suppose that the adversary has at least one server other than its ℓ lightest servers occupying some point in P at the time $\text{strategy}(\ell, P)$ is called. Then the adversary is able to serve all requests given in this call with cost at most c_ℓ by moving only its ℓ lightest servers.

Proof. We prove the claim by induction on ℓ . For the base case, suppose $\ell = 0$. Then $|P| = 1$ and by assumption, the adversary has at least one server at the unique point in P . Therefore, the adversary can serve the unique request given by $\text{strategy}(0, P)$ with cost $c_0 = 0$, without moving any server.

For the inductive case, suppose $\ell > 0$. We have assumed that the adversary has at least one server other than its lightest ℓ servers occupying some point p in P . By the third property of the set-system \mathcal{Q}_ℓ from Lemma 4, there exists a point $q \in P$ such that each set in \mathcal{Q}_ℓ contains at least one of p and q . The adversary moves its ℓ 'th lightest server to such a point q and keeps it there until the end of the $\text{strategy}(\ell, P)$ call. Due to this movement, the adversary incurs cost $\beta^{\ell-1}$, the first term in the definition of c_ℓ . As a result, both p and q become occupied by the adversary's servers other than the $\ell - 1$ lightest ones. We now show how the requests in all recursive calls made by $\text{strategy}(\ell, P)$ can be served by moving the $\ell - 1$ lightest servers only.

Consider any of the recursive calls made by $\text{strategy}(\ell, P)$. The set $P' \in \mathcal{Q}_\ell$ on which this call is made contains at least one of p and q . Both p and q were occupied by the adversary's servers other than the $\ell - 1$ lightest ones before $\text{strategy}(\ell, P)$ made its first recursive call. All the previous recursive calls were served by moving only the $\ell - 1$ lightest servers. Thus, at the time the current recursive call $\text{strategy}(\ell - 1, P')$ is made, points p and q are still occupied by the adversary's servers other than the $\ell - 1$ lightest ones. Therefore, at least one of these servers occupies a point in P' . By induction hypothesis, the adversary can serve all requests in the current recursive call $\text{strategy}(\ell - 1, P')$ with cost at most $c_{\ell-1}$ by moving only the $\ell - 1$ lightest servers. Since the number of such recursive calls is $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1) \leq \beta \cdot (\lceil n_{\ell-1}/2 \rceil + 1)$, the adversary serves all requests made in these calls with cost at most $\beta \cdot (\lceil n_{\ell-1}/2 \rceil + 1) \cdot c_{\ell-1}$, the second term in the expression for c_ℓ . \blacktriangleleft

We now use Corollary 6 and Lemma 7 to prove Theorem 2.

► **Theorem 2.** *The randomized competitive ratio of weighted k -server on uniform metric spaces is at least $H(n_{k-1}) = \Omega(2^k)$.*

Proof. We track the costs incurred by the algorithm and the adversary per $\text{strategy}(k-1, P)$ call made by the procedure **adversary**, and show that the former is at least $H(n_{k-1})$ times the latter.

Here is how the adversary serves the requests. Let q denote the position of the adversary's heaviest server at the time a $\text{strategy}(k-1, P)$ call is made. If $P = S \setminus \{q\}$, that is, the random point sampled from S turns out to be q , then the adversary finds the point q' which is sampled farthest in future by the procedure **adversary**, and moves its heaviest server there. These are the only movements of the adversary's heaviest server. By the standard coupon-collector argument, the expected number of samples from the current sample of q to q' is $(n_{k-1} + 1)H(n_{k-1})$, because $|S| = n_{k-1} + 1$ and we have already sampled q . Thus, in the long run, the cost of the adversary resulting from moving its heaviest server, per strategy call made by the procedure **adversary**, is $\beta^{k-1}/((n_{k-1} + 1)H(n_{k-1}))$.

By moving its heaviest server as described above, the adversary ensures the following. Before the adversary starts serving requests given by a $\text{strategy}(k-1, S \setminus \{p\})$ call, its heaviest server is located at some point different from p , and therefore, in $S \setminus \{p\}$. By Lemma 7, the adversary is able to serve requests given by each $\text{strategy}(k-1, S \setminus \{p\})$ with cost at most c_{k-1} without moving its heaviest server. In other words, the contribution of the adversary's servers other than the heaviest towards its cost per strategy call is at most c_{k-1} .

Thus, the adversary's cost per strategy call made by the procedure **adversary** is at most $\beta^{k-1}/((n_{k-1} + 1) \cdot H(n_{k-1})) + c_{k-1}$, which, by unrolling the recurrence in the statement of Lemma 7, is given by

$$\frac{\beta^{k-1}}{(n_{k-1} + 1) \cdot H(n_{k-1})} + c_{k-1} = \frac{\beta^{k-1}}{(n_{k-1} + 1) \cdot H(n_{k-1})} + \beta^{k-2} \cdot \sum_{i=1}^{k-1} \prod_{j=i}^{k-2} \left(\left\lceil \frac{n_j}{2} \right\rceil + 1 \right).$$

Let ε be an arbitrarily small positive number. By choosing

$$\beta = \lceil \varepsilon^{-1} \rceil \cdot (n_{k-1} + 1) \cdot H(n_{k-1}) \cdot \sum_{i=1}^{k-1} \prod_{j=i}^{k-2} \left(\left\lceil \frac{n_j}{2} \right\rceil + 1 \right),$$

the adversary's cost per strategy call is bounded from above by

$$\frac{\beta^{k-1} \cdot (1 + \varepsilon)}{(n_{k-1} + 1) \cdot H(n_{k-1})}.$$

On the other hand, recall from Corollary 6 that the expected cost of the algorithm per strategy call made by the procedure **adversary** is at least

$$\frac{(\beta - 1)^{k-1}}{n_{k-1} + 1} = \frac{\beta^{k-1}}{n_{k-1} + 1} \cdot \left(1 - \frac{1}{\beta}\right)^{k-1} \geq \frac{\beta^{k-1}}{n_{k-1} + 1} \cdot \left(1 - \frac{k-1}{\beta}\right) \geq \frac{\beta^{k-1} \cdot (1 - \varepsilon)}{n_{k-1} + 1},$$

because $\beta \gg k/\varepsilon$. Thus, modulo the $(1 \pm \varepsilon)$ factors, the algorithm's cost per strategy call is at least $H(n_{k-1})$ times the adversary's cost per strategy call. Since ε is arbitrarily small, we use Proposition 3 to conclude that the competitive ratio of any randomized online algorithm for weighted k -server on uniform metrics is at least $H(n_{k-1})$. ◀

4 Concluding Remarks

Given our lower bound on the randomized competitive ratio of weighted k -server on uniform metric spaces, the gap between the known upper and lower bounds has reduced from three orders of exponentiation to one. The natural question that needs to be investigated is to determine the randomized competitive ratio, or at least, prove upper and lower bounds that match in the order of exponentiation.

Our result also sheds light on the randomized competitive ratio of a generalization of the weighted k -server problem on uniform metrics called the generalized k -server problem on weighted uniform metrics. In this problem k servers are restricted to move in k different uniform metric spaces that are scaled copies of one another. A request contains one point from each copy and to serve it, one of the points must be covered by the server moving in its copy. Our lower bound directly applies to the generalized k -server problem on weighted uniform metrics and improves the previously known lower bound² of $\Omega(k/\log^2 k)$ by Bansal et al. [3] to exponential in k . This also proves that the generalized k -server problem on weighted uniform metrics is qualitatively harder than its unweighted counterpart, the generalized k -server problem on uniform metrics, which has randomized competitive ratio $O(k^2 \log k)$ due to Beinkowski, Jeż, and Schmidt [4].

References

- 1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.
- 2 Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k -server bounds via combinatorial dichotomies. In *FOCS*, pages 493–504, 2017. doi:10.1109/FOCS.2017.52.
- 3 Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k -server in uniform metrics. In *SODA*, pages 992–1001, 2018. doi:10.1137/1.9781611975031.64.
- 4 Marcin Bienkowski, Łukasz Jeż, and Paweł Schmidt. Slaying hydrae: Improved bounds for generalized k -server in uniform metrics. In *ISAAC*, volume 149 of *LIPIcs*, pages 14:1–14:14, 2019. doi:10.4230/LIPIcs.ISAAC.2019.14.
- 5 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k -server via multiscale entropic regularization. In *STOC*, pages 3–16, 2018. doi:10.1145/3188745.3188798.
- 6 Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k -server problems. *ACM Trans. Algorithms*, 16(1):14:1–14:28, 2020. doi:10.1145/3365002.
- 7 Marek Chrobak and Jiří Sgall. The weighted 2-server problem. *Theoretical Computer Science*, 324(2-3):289–312, 2004. doi:10.1016/j.tcs.2004.05.020.
- 8 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.
- 9 Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theoretical Computer Science*, 130(1):85–99, 1994. doi:10.1016/0304-3975(94)90154-6.
- 10 Elias Koutsoupias and Christos H. Papadimitriou. On the k -server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.
- 11 James R. Lee. Fusible HSTs and the randomized k -server conjecture. In *FOCS*, pages 438–449, 2018. doi:10.1109/FOCS.2018.00049.
- 12 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988. doi:10.1145/62212.62243.
- 13 Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. doi:10.1007/BF01759073.
- 14 René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014. doi:10.1137/120885309.
- 15 Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002. doi:10.1007/s00453-001-0124-5.

² This bound, in fact, holds for the unweighted counterpart, and to the best of the authors’ knowledge, no better bound for the weighted problem was known.

A Set-system Construction

► **Lemma 4.** *Let $\ell \in \mathbb{N}$ and let P be a set of n_ℓ points. There exists a set-system $\mathcal{Q}_\ell \subseteq 2^P$ satisfying the following properties.*

1. \mathcal{Q}_ℓ contains $\lceil n_{\ell-1}/2 \rceil + 1$ sets, each of size $n_{\ell-1}$.
2. For every $p \in P$, there exists a set in \mathcal{Q}_ℓ not containing p .
3. For every $p \in P$, there exists a $q \in P$ such that every set in \mathcal{Q}_ℓ contains at least one of p and q .

Proof (Bansal et al. [2]). Construct the set-system \mathcal{Q}_ℓ as follows. Recall that

$$|P| = n_\ell = \left(\left\lceil \frac{n_{\ell-1}}{2} \right\rceil + 1 \right) \cdot \left(\left\lfloor \frac{n_{\ell-1}}{2} \right\rfloor + 1 \right).$$

Let M be an arbitrary subset of P having size $\lceil n_{\ell-1}/2 \rceil + 1$, so that

$$|P \setminus M| = \left(\left\lceil \frac{n_{\ell-1}}{2} \right\rceil + 1 \right) \cdot \left\lfloor \frac{n_{\ell-1}}{2} \right\rfloor.$$

Partition $P \setminus M$ into $\lceil n_{\ell-1}/2 \rceil + 1$ sets of size $\lfloor n_{\ell-1}/2 \rfloor$ each, and for each $r \in M$, name a distinct set in the partition P'_r . Next, for each $r \in M$, define $P_r = (M \setminus \{r\}) \cup P'_r$, and let $\mathcal{Q}_\ell = \{P_r \mid r \in M\}$.

We now prove that \mathcal{Q}_ℓ indeed satisfies the required properties. First, the number of sets in \mathcal{Q}_ℓ is equal to $|M| = \lceil n_{\ell-1}/2 \rceil + 1$, and the size of each set $P_r \in \mathcal{Q}_\ell$ is

$$|P_r| = |M| - 1 + |P'_r| = \left\lceil \frac{n_{\ell-1}}{2} \right\rceil + \left\lfloor \frac{n_{\ell-1}}{2} \right\rfloor = n_{\ell-1}.$$

For the second property, observe that a point $p \in M$ is not contained in the corresponding set $P_p \in \mathcal{Q}_\ell$, whereas for a point $p \in P'_r$, the only set in \mathcal{Q}_ℓ that contains p is P_r . For the third property, if $p \in M$, define q to be any other point in M , and if $p \in P'_r$, define $q = r$, and check that the property is indeed satisfied. ◀

B Analysis of the Algorithm's Movements

We present the proof of Lemma 5 here, for which we need the following lemma.

► **Lemma 8.** *Let Z_1 and Z_2 be non-negative random variables and E be an event on a common sample space such that $\mathbb{E}[Z_1 \mid E] \geq b$ and $\mathbb{E}[Z_2 \mid \neg E] \geq b$ for some real number b . Then $\mathbb{E}[Z_1 + Z_2] \geq b$.*

Proof. We have,

$$\mathbb{E}[Z_1 + Z_2] = \mathbb{E}[Z_1 + Z_2 \mid E] \cdot \Pr[E] + \mathbb{E}[Z_1 + Z_2 \mid \neg E] \cdot \Pr[\neg E].$$

Since Z_1 and Z_2 are non-negative, we have,

$$\mathbb{E}[Z_1 + Z_2] \geq \mathbb{E}[Z_1 \mid E] \cdot \Pr[E] + \mathbb{E}[Z_2 \mid \neg E] \cdot \Pr[\neg E] \geq b \cdot \Pr[E] + b \cdot \Pr[\neg E] = b,$$

as required. ◀

► **Lemma 5.** *For every $\ell \in \{0, \dots, k-1\}$ the following holds. Let ρ_0 be an arbitrary sequence of requests and L be the set of positions of the algorithm's heaviest $k-\ell$ servers after serving ρ_0 . Let P be an arbitrary set of n_ℓ points disjoint from L . Suppose ρ_0 is followed by a random sequence ρ of requests given by a $\text{strategy}(\ell, P)$ call. For $i = 1, \dots, k$, let the random*

9:10 The Randomized Competitive Ratio of Weighted k -Server Is at Least Exponential

variable X_i denote the number of movements of the algorithm's i 'th lightest server while the algorithm serves ρ . Then we have,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell)} \cdot \mathbb{E}[X_i] \geq (\beta - 1)^\ell.$$

Proof. We prove the claim by induction on ℓ . For the base case, suppose $\ell = 0$. Then $|P| = 1$, and we are assured that L , the set of points occupied by all the algorithm's servers, is disjoint from P . In other words, none of the algorithm's servers occupies the unique point in P . Therefore, to serve the one request given by $\text{strategy}(0, P)$, the algorithm must move at least one of its servers, and thus,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell)} \cdot \mathbb{E}[X_i] = \sum_{i=1}^k \mathbb{E}[X_i] \geq 1 = (\beta - 1)^\ell,$$

as required.

For the inductive case, suppose $\ell > 0$. We are assured that except for the lightest ℓ servers, none of the servers of the algorithm occupies any point in P at the time the $\text{strategy}(\ell, P)$ call is made. This call makes $m = (\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1) = (\beta - 1) \cdot |\mathcal{Q}_\ell|$ recursive calls. For $i = 1, \dots, k$ and $j = 1, \dots, m$, let the random variable Y_i^j denote the number of movements of the algorithm's i 'th lightest server to serve requests from the j 'th recursive call. Thus, for all i , $X_i = \sum_{j=1}^m Y_i^j$.

Consider an arbitrary $j \in \{1, \dots, m\}$. Let E_j denote the event that the random variables $Y_i^{j'}$ are all 0 for all $i > \ell$ and $j' < j$. In words, E_j is the event that none of the algorithm's heaviest $k - \ell$ servers moves during the first $j - 1$ recursive calls. Recall that originally these servers did not occupy any point in P . Therefore, if E_j happens, these servers are guaranteed to be out of the set $P' \subseteq P$ on which the j 'th recursive call is made. Next, let E'_j denote the event that the j 'th recursive call is made on a set P' that does not contain the position of the algorithm's ℓ 'th lightest server after the first $j - 1$ recursive calls. Thus, if both E_j and E'_j happen, then P' is disjoint from the set of positions of the algorithm's $k - \ell + 1$ heaviest servers. We can then apply the induction hypothesis to get,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell-1)} \cdot \mathbb{E}[Y_i^j \mid E_j \wedge E'_j] \geq (\beta - 1)^{\ell-1}. \quad (1)$$

Next, let us understand the behavior of the random variables Y_i^j conditioned on E_j only. We have,

$$\mathbb{E}[Y_i^j \mid E_j] \geq \mathbb{E}[Y_i^j \mid E_j \wedge E'_j] \cdot \Pr[E'_j \mid E_j] \geq \frac{\mathbb{E}[Y_i^j \mid E_j \wedge E'_j]}{|\mathcal{Q}_\ell|}. \quad (2)$$

Here, the first inequality holds because Y_i^j is non-negative. The second inequality holds because, by the second property of the set-system \mathcal{Q}_ℓ given by Lemma 4, for every possible history before the j 'th recursive call, \mathcal{Q}_ℓ contains at least one set which does not contain the position of the algorithm's ℓ 'th lightest server. This implies $\Pr[E'_j \mid E_j] \geq 1/|\mathcal{Q}_\ell|$. From Equation 1 and Equation 2, we get,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell-1)} \cdot \mathbb{E}[Y_i^j \mid E_j] \geq \frac{(\beta - 1)^{\ell-1}}{|\mathcal{Q}_\ell|}. \quad (3)$$

By the non-negativity of the random variables $Y_i^{j'}$ and the definition of E_j , we trivially have,

$$\sum_{i=\ell+1}^k \sum_{j'=1}^m \mathbb{E}[Y_i^{j'} \mid \neg E_j] \geq \sum_{i=\ell+1}^k \sum_{j'=1}^{j-1} \mathbb{E}[Y_i^{j'} \mid \neg E_j] \geq 1,$$

and hence,

$$\frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|} \cdot \sum_{i=\ell+1}^k \sum_{j'=1}^m \mathbb{E}[Y_i^{j'} \mid \neg E_j] \geq \frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|}. \quad (4)$$

From Equation 3 and Equation 4, using Lemma 8, we get,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell-1)} \cdot \mathbb{E}[Y_i^j] + \frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|} \cdot \sum_{i=\ell+1}^k \sum_{j'=1}^m \mathbb{E}[Y_i^{j'}] \geq \frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|}.$$

The above inequality holds for all $j \in \{1, \dots, m\}$. Summing up over all j and recalling $X_i = \sum_{j=1}^m Y_i^j$, we get,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell-1)} \cdot \mathbb{E}[X_i] + m \cdot \frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|} \cdot \sum_{i=\ell+1}^k \mathbb{E}[X_i] \geq m \cdot \frac{(\beta-1)^{\ell-1}}{|\mathcal{Q}_\ell|}.$$

Recall that $m = (\beta-1) \cdot |\mathcal{Q}_\ell|$. Thus,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell-1)} \cdot \mathbb{E}[X_i] + (\beta-1)^\ell \cdot \sum_{i=\ell+1}^k \mathbb{E}[X_i] \geq (\beta-1)^\ell. \quad (5)$$

Finally, note that for $i > \ell$, $\min(i-1, \ell-1) = \ell-1$, and since $\ell \geq 1$, we have $(\beta-1)^\ell \leq \beta^{\ell-1}(\beta-1)$. Therefore, the multiplier of the $\mathbb{E}[X_i]$ term in Equation 5 is bounded as,

$$\beta^{\min(i-1, \ell-1)} + (\beta-1)^\ell \leq \beta^{\ell-1} + \beta^{\ell-1}(\beta-1) = \beta^\ell = \beta^{\min(i-1, \ell)}.$$

On the other hand, for $i \leq \ell$, $\min(i-1, \ell-1) = \min(i-1, \ell)$. Thus, for all $i \in \{1, \dots, k\}$, the multiplier of the $\mathbb{E}[X_i]$ term in Equation 5 is at most $\beta^{\min(i-1, \ell)}$. Since the X_i 's are all non-negative, we get,

$$\sum_{i=1}^k \beta^{\min(i-1, \ell)} \cdot \mathbb{E}[X_i] \geq (\beta-1)^\ell,$$

as required. ◀

Orienting (Hyper)graphs Under Explorable Stochastic Uncertainty

Evrpidis Bampis ✉ 🏠 

Sorbonne Université, CNRS, LIP6, Paris, France

Christoph Dürr ✉ 🏠 

Sorbonne Université, CNRS, LIP6, Paris, France

Thomas Erlebach ✉ 🏠 

School of Informatics, University of Leicester, UK

Murilo Santos de Lima ✉ 🏠 

School of Informatics, University of Leicester, UK

Nicole Megow ✉ 🏠 

Faculty of Mathematics and Computer Science, University of Bremen, Germany

Jens Schlöter ✉ 🏠 

Faculty of Mathematics and Computer Science, University of Bremen, Germany

Abstract

Given a hypergraph with uncertain node weights following known probability distributions, we study the problem of querying as few nodes as possible until the identity of a node with minimum weight can be determined for each hyperedge. Querying a node has a cost and reveals the precise weight of the node, drawn from the given probability distribution. Using competitive analysis, we compare the expected query cost of an algorithm with the expected cost of an optimal query set for the given instance. For the general case, we give a polynomial-time $f(\alpha)$ -competitive algorithm, where $f(\alpha) \in [1.618 + \epsilon, 2]$ depends on the approximation ratio α for an underlying vertex cover problem. We also show that no algorithm using a similar approach can be better than 1.5-competitive. Furthermore, we give polynomial-time $4/3$ -competitive algorithms for bipartite graphs with arbitrary query costs and for hypergraphs with a single hyperedge and uniform query costs, with matching lower bounds.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases Explorable uncertainty, queries, stochastic optimization, graph orientation, selection problems

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.10

Related Version *Full Version:* <https://arxiv.org/abs/2107.00572>

Funding *Evrpidis Bampis:* Supported by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

Christoph Dürr: Supported by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

Thomas Erlebach: Supported by EPSRC grant EP/S033483/1.

Murilo Santos de Lima: Funded by EPSRC grant EP/S033483/1.

Nicole Megow: Supported by the German Science Foundation (DFG) under contract ME 3825/1.

Jens Schlöter: Funded by the German Science Foundation (DFG) under contract ME 3825/1.

Acknowledgements The authors would like to thank the anonymous referees for their careful reading and helpful suggestions.



© Evripidis Bampis, Christoph Dürr, Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter;

licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 10; pp. 10:1–10:18



Leibniz International Proceedings in Informatics

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

1 Introduction

The research area of explorable uncertainty is concerned with scenarios where parts of the input are initially uncertain, but the precise weight (or value) of an input item can be obtained via a *query*. For example, an uncertain weight may be represented as an interval that is guaranteed to contain the precise weight, but only a query can reveal the precise weight. Adaptive algorithms make queries one by one until they have gathered sufficient information to solve the given problem. The goal is to make as few queries as possible. In most of the previous work on explorable uncertainty, an adversarial model has been studied where an adversary determines the precise weights of the uncertain elements in such a way that the performance of the algorithm, compared to the optimal query set, is as bad as possible. While this model provides worst-case guarantees that hold for every possible instance, it is also very pessimistic because the adversary is free to choose the precise weights arbitrarily. In realistic scenarios, one may often have some information about where in the given interval the precise weight of an uncertain element is likely to lie. This information can be represented as a probability distribution and exploited in order to achieve better performance guarantees.

In this paper, we study the following problem under stochastic uncertainty: Given a family of (not necessarily disjoint) subsets of a set of uncertain elements, determine the element with minimum precise weight in each set, using queries of minimum total cost. Note that we do not necessarily need to obtain the precise minimum weight. We phrase the problem in the language of hypergraphs, where each uncertain element corresponds to a node and each set corresponds to a hyperedge. We call this the *hypergraph orientation* problem, as we can think of orienting each hyperedge towards its minimum-weight vertex. Each node $v \in V$ of a hypergraph $H = (V, E)$ is associated with a known continuous probability distribution¹ d_v over an interval $I_v = (\ell_v, r_v)$ and has query cost c_v . The precise weight of v is drawn independently from d_v and denoted by w_v . We assume that I_v is the minimal interval that contains the support of d_v , i.e., ℓ_v is the largest value satisfying $\mathbb{P}[w_v \leq \ell_v] = 0$ and r_v is the smallest value satisfying $\mathbb{P}[w_v \geq r_v] = 0$. For $S \subseteq V$, we define $c(S) = \sum_{v \in S} c_v$. An algorithm can sequentially make queries to vertices to learn their weights, until it has enough information to identify the minimum-weight vertex of each hyperedge. A query of v reveals its precise weight w_v , which is drawn independently from d_v . If all vertices have the same query cost, we say that the query costs are uniform and assume w.l.o.g. that $c_v = 1$ for all $v \in V$. Otherwise, we speak of arbitrary query costs. The objective of an algorithm is to minimize the expected cost of the queries it makes.

We also consider the special case where we are given a graph $G = (V, E)$ instead of a hypergraph $H = (V, E)$, called the *graph orientation* problem.

As an example consider a multi-national medical company that needs a certain product (say, chemical ingredient, medicine or vaccine) for its operation in each country. The particular products that are available in each country are different due to different approval mechanisms. The task is to find the best product for each country, that is, the best among the approved ones. The quality itself is independent of the country and can be determined by extensive tests in a lab (queries). The set of products available in one country corresponds to a hyperedge, and the problem of identifying the best product in each country is the hypergraph orientation problem.

¹ We assume the distribution is given in such a way that $\mathbb{P}[w_v \in (a, b)]$ can be computed in polynomial time for every $v \in V, a, b \in \mathbb{R}$. For all our algorithms it suffices to be given a probability matrix: rows correspond to vertices v , columns to elementary intervals (t_i, t_{i+1}) , and entries to $\mathbb{P}[w_v \in (t_i, t_{i+1})]$, where $t_1, \dots, t_{2|V|}$ represent the sorted elements of $\{\ell_v, r_v | v \in V\}$.

Our contribution. Our main result (Section 3) is an algorithm for the graph orientation problem with competitive ratio $\frac{1}{2}(\alpha + \sqrt{8 - \alpha(4 - \alpha)})$, assuming we have an α -approximation for the vertex cover problem (which we need to solve on an induced subgraph of the given graph). This factor is always between $\phi \approx 1.618$ (for $\alpha = 1$), and 2 (for $\alpha = 2$). We show that, for the special cases of directing $\mathcal{O}(\log |V|)$ hyperedges and sorting $\mathcal{O}(1)$ sets, the algorithm can be applied with $\alpha = 1$ in polynomial running time. The algorithm has a preprocessing phase in two steps. First, we compute the probability that a vertex is *mandatory*, i.e., that it is part of any feasible solution, and we query all vertices with probability over a certain threshold. The second step uses a LP relaxation of the vertex cover problem to select some further vertices to query. Next, we compute an α -approximation of the vertex cover on a subgraph induced by the preprocessing, and we query the vertices in the given solution. The algorithm finishes with a postprocessing that only queries mandatory intervals. For the analysis, we show two main facts: (1) the expected optimal solution can be bounded by the expected optimal solutions for the subproblems induced by a partition of the vertices; (2) for the subproblem on which we compute a vertex cover, the expected optimal solution can be bounded by applying the König-Egerváry theorem [52] on a particular bipartite graph, in case of uniform costs. When given arbitrary query costs, we show in the full version [6] that we can utilize a technique of *splitting* the vertices in order to obtain a collection of disjoint stars with obvious vertex covers that imply a bound on the expected optimum.

We further show how to generalize the algorithm to hypergraphs. Unfortunately in this case it is $\#P$ -hard to compute the probability of a vertex being mandatory, but we can approximate it by sampling. This yields a randomized algorithm that attains, with high probability, a competitive ratio arbitrarily close to the expression given above for graphs. Here, we need to solve the vertex cover problem on an induced subgraph of an auxiliary graph that contains, for each hyperedge of the given hypergraph, all edges between the node with the leftmost interval and the nodes whose intervals intersect that interval.

We also consider a natural alternative algorithm (Section 4) that starts with a particular vertex cover solution followed by adaptively querying remaining vertices. We prove a competitive ratio of $4/3$ on special cases, namely, for bipartite graphs with arbitrary cost and for a single hyperedge with uniform costs, and complement this by matching lower bounds.

Related work. Graph orientation problems are fundamental in the area of graph theory and combinatorial optimization. In general, graph orientation refers to the task of giving an orientation to edges in an undirected graph such that some given requirement is met. Different types of requirements have been investigated. While Robbins [50] initiated research on connectivity and reachability requirements already in the 1930s, most work is concerned with degree-constraints; cf. overviews given by Schrijver [52, Chap. 61] and Frank [29, Chap. 9].

Our requirement, orienting each edge towards its node with minimum weight, becomes challenging when there is uncertainty in the node weights. While there are different ways of modeling uncertainty in the input data, the model of explorable uncertainty was introduced by Kahan [40]. He considers the task of identifying the minimum element in a set of uncertainty intervals, which is equivalent to orienting a single hyperedge. Unlike in our model, no distributional information is known, and an adversary can choose weights in a worst-case manner from the intervals. Kahan [40] shows that querying the intervals in order of non-decreasing left endpoints requires at most one more query than the optimal query set, thus giving a competitive ratio of 2. Further, he shows that this is best possible in the adversarial model.

Subsequent work addresses finding the k -th smallest value in a set of uncertainty intervals [27,37], caching problems [49], computing a function value [41], and classical combinatorial optimization problems, such as shortest path [26], knapsack [31], scheduling problems [2,3,21], minimum spanning tree and matroids [22,25,28,45,46]. Recent work on sorting elements of a single or multiple non-disjoint sets is particularly relevant as it is a special case of the graph orientation problem [24,38]. For sorting a single set in the adversarial explorable uncertainty model, there is a 2-competitive algorithm and it is best possible, even for arbitrary query costs [38]. The competitive ratio can be improved to 1.5 for uniform query cost by using randomization [38]. Algorithms with limited adaptivity have been proposed in [23].

Although the adversarial model is arguably pessimistic and real-world applications often come with some distributional information, surprisingly little is known on stochastic variants of explorable uncertainty. The only previous work we are aware of is by Chaplick et al. [16], in which they studied stochastic uncertainty for the problem of sorting a given set of uncertain elements, and for the problem of determining the minimum element in a given set of uncertain elements. They showed that the optimal decision tree (i.e., an algorithm that minimizes the expected query cost among all algorithms) for a given instance of the sorting problem can be computed in polynomial time. For the minimum problem, they leave open whether an optimal decision tree can be determined in polynomial time, but give a 1.5-competitive algorithm and an algorithm that guarantees a bound slightly smaller than 1.5 on the expectation of the ratio between the query cost of the algorithm and the optimal query cost. The problem of scheduling with testing [42] is also in the spirit of stochastic explorable uncertainty but less relevant here.

There are many other stochastic problems that take exploration cost into account. Some of the earliest work has studied multi-armed bandits [15,30,54] and Weitzman's Pandora's box problem [55], which are prime examples for analyzing the tradeoff between the cost for exploration and the benefit from exploiting gained information. More recently, query-variants of combinatorial optimization problems received some attention, in general [33,53], and for specific problems such as stochastic knapsack [20,43], orienteering [8,34], matching [5,7,12,13,17], and probing problems [1,35,36]. Typically such work employs a *query-commit* model, meaning that queried elements must be part of the solution, or solution elements are required to be queried. These are quite strong requirements that lead to a different flavor of the cost-benefit tradeoff.

Research involving queries to identify particular graph structures or elements, or queries to verify certain properties, can be found in various flavors. A well-studied problem class is *property testing* [32], and there are many more, see e.g., [4,11,18,44,48,51]. Without describing such problems in detail, we emphasize a fundamental difference to our work. Typically, in these query models, the bounds on the number of queries made by an algorithm are *absolute numbers*, i.e., given as a function of the input size, but independent of the input graph itself and without any comparison to the minimum number of queries needed for the given graph.

2 **Definitions and Preliminary Results**

The hypergraph orientation problem and the graph orientation problem have already been defined in Section 1. In this section we first give additional definitions and discuss how we measure the performance of an algorithm. Then we introduce the concept of mandatory vertices and show how the probability for a vertex to be mandatory can be computed or at least approximated efficiently. We also give a lower bound showing that no algorithm

can achieve competitive ratio better than $\frac{4}{3}$. Next, based on the concept of witness sets, we define the vertex cover instance associated with an instance of our problem and define a class of vertex cover-based algorithms, which includes all the algorithms we propose in this paper. Finally, we characterize the optimal query set for each realization and give lower bounds on the expected optimal query cost, which we will use later in the analysis of our algorithms.

Definitions. To measure the performance of an algorithm, we compare the expected cost of the queries it makes to the expected optimal query cost. Formally, given a realization of the values, we call *feasible query set* a set of vertices to be queried that permits one to identify the minimum-weight vertex in every hyperedge. Note that a query set is feasible if, for each hyperedge, it either queries the node v with minimum weight w_v and all other nodes whose intervals contain w_v , or it does not query the node v with minimum weight but queries all nodes whose intervals overlap I_v , and in addition the precise weights of all those intervals lie to the right of I_v . An *optimal query set* is a feasible query set of minimum query cost. We denote by $\mathbb{E}[\text{OPT}]$ the expected query cost of an optimal query set. Similarly, we denote by $\mathbb{E}[A]$ the expected query cost of the query set queried by an algorithm A . The supremum of $\mathbb{E}[A]/\mathbb{E}[\text{OPT}]$, over all instances of the problem, is called the *competitive ratio* of A . Alternatively, one could compare $\mathbb{E}[A]$ against the cost $\mathbb{E}[A^*]$ of an optimal adaptive algorithm A^* . However, in explorable uncertainty, it is standard to compare against the optimal query set, and, since $\mathbb{E}[\text{OPT}]$ is a lower bound on $\mathbb{E}[A^*]$, all our algorithmic results translate to this alternative setting.

Let $F \in E$ be a hyperedge consisting of vertices v_1, \dots, v_k , indexed in order of non-decreasing left endpoints of the intervals, i.e., $\ell_{v_1} \leq \dots \leq \ell_{v_k}$. We call v_1 the *leftmost* vertex of F . We can assume that $I_{v_1} \cap I_{v_i} \neq \emptyset$ for all $2 \leq i \leq k$, because otherwise the vertex v_i could be removed from the hyperedge F . For the special case of graphs, this means that we assume $I_v \cap I_u \neq \emptyset$ for each $\{u, v\} \in E$, since otherwise we could simply remove the edge.

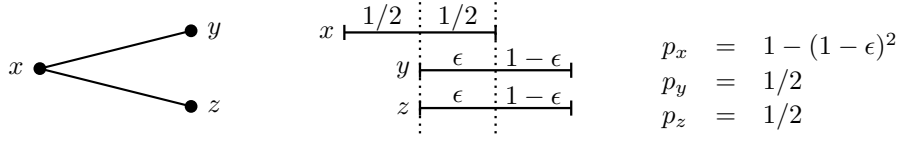
Mandatory vertices, probability to be mandatory. A vertex v is called *mandatory* if it belongs to every feasible query set for the given realization. For example, if for some edge $\{u, v\}$, vertex u has already been queried and its value w_u belongs to the interval I_v , then v is known to be mandatory. The following lemma was shown in [24] and fully characterizes mandatory vertices.

► **Lemma 2.1.** *A vertex $v \in V$ is mandatory if and only if there is a hyperedge $F \in E$ with $v \in F$ such that either (i) v is a minimum-weight vertex of F and $w_u \in I_v$ for some $u \in F \setminus \{v\}$, or (ii) v is not a minimum-weight vertex of F and $w_u \in I_v$ for the minimum-weight vertex u of F .*

For a hyperedge $F = \{v_1, \dots, v_k\}$, where the vertices are again indexed by non-decreasing left endpoints, it was shown in [16, Section 3] that, if $I_{v_i} \subseteq I_{v_1}$ for some $2 \leq i \leq k$, then v_1 is mandatory for every realization. Thus, every algorithm can iteratively query all such elements in a preprocessing step, without worsening the competitive ratio. In the remainder of the paper, we assume w.l.o.g. that the instance under consideration is already preprocessed.

Similarly, if a hyperedge contains vertices u, v such that v has not been queried yet and is the leftmost vertex, while a query of u has revealed that $w_u \in I_v$, then it follows from Lemma 2.1 that v is mandatory for every realization of the unqueried vertices. The final stage of our algorithms will consist of querying mandatory vertices that are identified by this criterion, until the instance is solved.

We denote by p_v the probability that a vertex v is mandatory. Querying vertices $v \in V$ that have a high probability p_v is a key element of our algorithms. For graphs, p_v is easy to compute as, by Lemma 2.1, v is mandatory iff $w_u \in I_v$ for some neighbor vertex u .



■ **Figure 1** Instance and mandatory probabilities used in the proof of Theorem 2.3.

Hence, $p_v = 1 - \prod_{u:\{u,v\} \in E} \mathbb{P}[w_u \notin I_v]$. For hypergraphs, however, we can show that the computation of p_v is $\#P$ -hard, even if all hyperedges have size 3. Luckily it is not difficult to get a good estimate of the probabilities to be mandatory for hypergraphs using sampling.

► **Lemma 2.2.** *There is a polynomial-time randomized algorithm that, given a hypergraph $H = (V, E)$, a vertex $v \in V$, and parameters $\epsilon, \delta \in (0, 1)$, produces a value y such that $|y - p_v| \geq \epsilon$ with probability at most δ . Its time complexity is $O(|V| \ln(1/\delta)/\epsilon^2)$.*

General lower bound. We have the following lower bound.

► **Theorem 2.3.** *Every algorithm for the graph orientation problem has competitive ratio at least $\frac{4}{3}$, even for uniform query costs and even if no restriction on the running time of the algorithm is imposed.*

Proof. Consider three vertices x, y, z , with $I_x = (0, 2)$ and $I_y = I_z = (1, 3)$, and uniform query costs $c_x = c_y = c_z = 1$. The only edges are $\{x, y\}$ and $\{x, z\}$. The probabilities are such that $\mathbb{P}[w_x \in (1, 2)] = \frac{1}{2}$ and $\mathbb{P}[w_y \in (1, 2)] = \mathbb{P}[w_z \in (1, 2)] = \epsilon$, for some $0 < \epsilon \ll \frac{1}{2}$; see Figure 1. If $w_x \in (0, 1]$, which happens with probability $\frac{1}{2}$, querying x is enough. If $w_x \in (1, 2)$ and $w_y, w_z \in [2, 3]$, which happens with probability $\frac{1}{2}(1 - \epsilon)^2$, querying y and z is enough. Otherwise, all three vertices must be queried. We have

$$\mathbb{E}[\text{OPT}] = \frac{1}{2} \cdot 1 + \frac{1}{2}(1 - \epsilon)^2 \cdot 2 + \frac{1}{2} \left(1 - (1 - \epsilon)^2\right) \cdot 3 = 2 - \frac{(1 - \epsilon)^2}{2},$$

which tends to $\frac{3}{2}$ as ϵ approaches 0. Since y and z are identical and we can assume that an algorithm always queries first a vertex that it knows to be mandatory (if there is one), we only have three possible decision trees to consider:

1. First query x ; if $w_x \in (1, 2)$, then query y and z . The expected query cost is 2.
2. First query y . If $w_y \in (1, 2)$, then query x , and query z if $w_x \in (1, 2)$. If $w_y \in [2, 3]$, then query z , and query x if $w_z \in (1, 2)$. The expected query cost is $1 + \frac{3}{2}\epsilon + (1 - \epsilon)(1 + \epsilon)$, which tends to 2 as ϵ approaches 0.
3. First query y . Whatever happens, query x , then query z if $w_x \in (1, 2)$. The expected query cost is $\frac{5}{2}$, so this is never better than the previous options.

With either choice (even randomized), the competitive ratio tends to at least $\frac{4}{3}$ as $\epsilon \rightarrow 0$. ◀

This lower bound can be adapted for a single hyperedge $\{x, y, z\}$. For arbitrary query costs, it works even for a single edge $\{x, y\}$, by taking $c_x = 1$ and $c_y = 2$.

Witness sets, vertex cover instance, vertex cover-based algorithms. Another key concept of our algorithms is to exploit *witness sets* [14, 25]. A subset $W \subseteq V$ is a *witness set* if $W \cap Q \neq \emptyset$ for all feasible query sets Q . The following lemma was shown in [40].

► **Lemma 2.4.** *Let $F = \{v_1, \dots, v_k\}$ be a hyperedge, and let v_1 be the leftmost vertex of F . Then $\{v_1, v_i\}$ is a witness set for each $2 \leq i \leq k$.*

The lemma implies that one can obtain a 2-competitive algorithm in the adversarial model for the hyperedge orientation problem: For uniform query costs, it suffices to repeatedly query witness sets of size 2 until the instance is solved, by a standard witness set argument [25]. For arbitrary query costs, this approach can be combined with the local ratio technique [9] to obtain the same competitive ratio (in a similar way as done in [38] for the sorting problem). Our goal is to achieve better competitive ratios in the stochastic setting. Motivated by Lemma 2.4, we can now define the *vertex cover instance*.

► **Definition 2.5.** *Given a hypergraph $H = (V, E)$, the vertex cover instance of H is the graph $\bar{G} = (V, \bar{E})$ with $\{v, u\} \in \bar{E}$ if and only if there is a hyperedge $F \in E$ such that $v, u \in F$, v is leftmost in F and $I_v \cap I_u \neq \emptyset$. For the special case of a graph G instead of a hypergraph H , it holds that $\bar{G} = G$.*

Since each edge of the vertex cover instance \bar{G} is a witness set by Lemma 2.4, we can observe that each feasible query set Q is a vertex cover of \bar{G} . Using the vertex cover instance, we can define a class of algorithms for the hypergraph orientation problem as follows: An algorithm is *vertex cover-based* if it implements the following pattern:

1. Non-adaptively query a vertex cover VC of \bar{G} ;
2. Iteratively query mandatory vertices until the minimum-weight vertex of each hyperedge is known: For each hyperedge $F \in E$ for which the minimum weight is still unknown, query the vertices in order of left endpoints until the minimum weight is found.

By definition of the second step, each vertex cover-based algorithm clearly orients each hyperedge. Furthermore, Lemma 2.1 implies that each vertex queried in the last step is indeed mandatory for all realizations that are consistent with the currently known information, i.e., the weights of the previously queried vertices. For graphs, this is easy to see, and for hypergraphs, this can be shown as follows: For a hyperedge F that isn't solved after the first step and has leftmost vertex v initially, the vertex cover VC has queried v or all other vertices of F . In the latter case, v is the only unqueried vertex of F and I_v must contain the precise weight of one of the other vertices, hence v is mandatory. In the former case, the remaining candidates for being the minimum-weight vertex are (1) the vertex with leftmost precise weight among those queried in the first step, and (2) the unqueried vertices whose intervals contain that precise weight. It is then clear that the leftmost vertex is mandatory, and querying it either solves the hyperedge or yields a situation of the same type.

All the algorithms we propose in this paper are vertex cover-based. We have the following lower bounds for vertex cover-based algorithms.

► **Theorem 2.6.** *No vertex cover-based algorithm has competitive ratio better than $\frac{3}{2}$ for the hypergraph orientation problem. This result holds even in the following special cases:*

1. *The graph has only a single hyperedge but the query costs are not uniform.*
2. *The query costs are uniform and the vertex cover instance \bar{G} is bipartite.*
3. *The instance is a non-bipartite graph orientation instance with uniform query costs.*

We remark that the second step of vertex cover-based algorithms must be adaptive: In the full version we show that any algorithm consisting of two non-adaptive stages cannot have competitive ratio $o(\log n)$, even for a single hyperedge with n vertices and uniform query costs.

Bounds on $\mathbb{E}[\text{OPT}]$. Let \mathcal{R} be the set of all possible realizations and let $\text{OPT}(R)$ for $R \in \mathcal{R}$ be the optimal query cost for realization R . As each feasible query set Q must include a vertex cover of \bar{G} , the minimum weight of a vertex cover of \bar{G} (using the query costs as

weights) is a lower bound on the optimal query cost for each realization and, thus, on $\mathbb{E}[\text{OPT}]$. This observation in combination with Lemma 2.1 also gives us a way to identify an optimal query set for a fixed realization, by using the knowledge of the exact vertex weights.

► **Observation 2.7.** *For a fixed realization R of an instance of the hypergraph orientation problem, let M be the set of vertices that are mandatory (cf. Lemma 2.1), and let VC_M be a minimum-weight vertex cover of $\bar{G}[V \setminus M]$. Then $M \cup VC_M$ is an optimal query set for realization R .*

Computing $\text{OPT}(R)$ for a fixed and known realization R is NP-hard [24]. This extends to the hypergraph orientation problem and the computation of $\mathbb{E}[\text{OPT}]$: We can reduce from the problem of computing $\text{OPT}(R)$ by concentrating the probability mass of all intervals onto the weights in realization R . The reduction of [24] in combination with [19] also implies APX-hardness.

To analyze the performance of our algorithms, we compare the expected cost of the algorithms to the expected cost of the optimal solution. By Observation 2.7, $c(M) + c(VC_M)$ is the minimum query cost for a fixed realization R , where $M \subseteq V$ is the set of mandatory elements in the realization and VC_M is a minimum-weight vertex cover for the subgraph $\bar{G}[V \setminus M]$ of the vertex cover instance $\bar{G} = (V, \bar{E})$ induced by $V \setminus M$. Thus, the optimal solution for a fixed realization is completely characterized by the set of mandatory elements in the realization. Using this, we can characterize $\mathbb{E}[\text{OPT}]$ as $\mathbb{E}[\text{OPT}] = \sum_{M \subseteq V} p(M) \cdot c(M) + \sum_{M \subseteq V} p(M) \cdot c(VC_M)$, where $p(M)$ denotes the probability that M is the set of mandatory elements. It follows that $\sum_{M \subseteq V} p(M) \cdot c(M) = \sum_{v \in V} p_v \cdot c(v)$, since both terms describe the expected cost for querying mandatory elements, which leads to the following characterization of $\mathbb{E}[\text{OPT}]$:

$$\mathbb{E}[\text{OPT}] = \sum_{v \in V} p_v \cdot c_v + \sum_{M \subseteq V} p(M) \cdot c(VC_M).$$

A key technique for our analysis is lower bounding $\mathbb{E}[\text{OPT}]$ by partitioning the optimal solution into subproblems and discarding dependencies between elements in different subproblems.

► **Definition 2.8.** *For a realization R and any subset $S \subseteq V$, let $\text{OPT}_S = \min_{Q \in \mathcal{Q}} c(Q \cap S)$, where \mathcal{Q} is the set of all feasible query sets for realization R .*

► **Lemma 2.9.** *Let S_1, \dots, S_k be a partition of V . Then $\mathbb{E}[\text{OPT}] \geq \sum_{i=1}^k \mathbb{E}[\text{OPT}_{S_i}]$.*

Proof. We start the proof by characterizing $\mathbb{E}[\text{OPT}_{S_i}]$ for each $i \in \{1, \dots, k\}$. Let $R \in \mathcal{R}$ be a realization in which M is the set of mandatory elements. Then OPT_{S_i} needs to contain all mandatory elements of S_i , and resolve all remaining dependencies between vertices of S_i , i.e., query a minimum-weight vertex cover $VC_M^{S_i}$ for the subgraph $\bar{G}[S_i \setminus M]$. Thus, it follows

$$\mathbb{E}[\text{OPT}_{S_i}] = \sum_{v \in S_i} p_v \cdot c_v + \sum_{M \subseteq V} p(M) \cdot c(VC_M^{S_i}). \quad (1)$$

By summing Equation (1) over all $i \in \{1, \dots, k\}$, we obtain the lemma:

$$\begin{aligned} \sum_{i=1}^k \mathbb{E}[\text{OPT}_{S_i}] &= \sum_{i=1}^k \left(\sum_{v \in S_i} p_v \cdot c_v + \sum_{M \subseteq V} p(M) \cdot c(VC_M^{S_i}) \right) \\ &= \sum_{v \in V} p_v \cdot c_v + \sum_{M \subseteq V} p(M) \cdot \left(\sum_{i=1}^k c(VC_M^{S_i}) \right) \\ &\leq \sum_{v \in V} p_v \cdot c_v + \sum_{M \subseteq V} p(M) \cdot c(VC_M) = \mathbb{E}[\text{OPT}], \end{aligned}$$

where the second equality follows from S_1, \dots, S_k being a partition. The inequality follows from $\sum_{i=1}^k c(VC_M^{S_i})$ being the cost of a minimum weighted vertex cover for a subgraph of $G[V \setminus M]$, while $c(VC_M)$ is the minimum cost for a vertex cover of the whole graph. ◀

For the case of arbitrary query costs, we will sometimes need to partition V in such a way that a vertex v can be split into fractions that are in different parts of the partition. We view each fraction as a copy of v , and the split is done in such a way that the query costs of all copies of v add up to c_v . Further, the probability distribution for being mandatory in the resulting instance is such that either all copies of v are mandatory or none of them is, and the former happens with probability p_v . (A detailed discussion of this process can be found in the full version.) We refer to the application of this operation to a vertex as a *vertex split* and note that it can be applied repeatedly.

► **Observation 2.10.** *Let OPT' be the optimal solution for an instance that is created by iteratively executing vertex splits. Then, $\mathbb{E}[\text{OPT}'] = \mathbb{E}[\text{OPT}]$. Furthermore, Lemma 2.9 also applies to $\mathbb{E}[\text{OPT}']$ and the modified instance.*

3 A Threshold Algorithm for Orienting Hypergraphs

We present an algorithm for orienting graphs and its generalization to hypergraphs.

3.1 Orienting Graphs

We consider the graph orientation problem. As a subproblem, we solve a vertex cover problem. This problem is NP-hard and 2-approximation algorithms are known [56]. For several special graph classes, there are improved algorithms [39]. Using an α -approximation as a black box, we give a competitive ratio between $\phi \approx 1.618$ ($\alpha = 1$) and 2 ($\alpha = 2$) as a function depending on α .

Algorithm 1 THRESHOLD.

Input: Instance $G = (V, E)$, p_v for each $v \in V$, parameter $d \in [0, 1]$,
and an α -approximation black box for the vertex cover problem

- 1 Let $M = \{v \in V \mid p_v \geq d\}$;
- 2 Solve (LP) for $G[V \setminus M]$ and let x^* be an optimal basic feasible solution;
- 3 Let $V_1 = \{v \in V \mid x_v^* = 1\}$ and similarly $V_{1/2}, V_0$;
- 4 Use the α -approximation black box to approximate a vertex cover VC' for $G[V_{1/2}]$;
- 5 Query $Q = M \cup V_1 \cup VC'$; /* Q is a vertex cover of G */
- 6 Query the mandatory elements of $V \setminus Q$;

10:10 Orienting (Hyper)graphs Under Explorable Stochastic Uncertainty

Algorithm 1 is parameterized by a threshold $d \in [0, 1]$, which is optimized depending on the approximation ratio α of the chosen vertex cover procedure. The algorithm executes a preprocessing of the vertex cover instance by using the following classical LP relaxation, for which each optimal basic feasible solution is half-integral [47]:

$$\begin{aligned} \min \quad & \sum_{v \in V} c_v \cdot x_v \\ \text{s.t.} \quad & x_v + x_u \geq 1 \quad \forall \{u, v\} \in E \\ & x_v \geq 0 \quad \forall v \in V \end{aligned} \tag{LP}$$

► **Theorem 3.1.** *Given an α -approximation with $1 \leq \alpha \leq 2$ for the vertex cover problem (on the induced subgraph $G[V_{1/2}]$, see Line 4), THRESHOLD with parameter d achieves a competitive ratio of $\max\{\frac{1}{d}, \alpha + (2 - \alpha) \cdot d\}$ for the graph orientation problem. Optimizing d yields a competitive ratio of $\frac{1}{2}(\alpha + \sqrt{8 - \alpha(4 - \alpha)})$.*

Proof. Here, we show the result for uniform query costs. The generalization to arbitrary query costs requires an additional technical step involving vertex splitting and is discussed in the full version. Since Q is a vertex cover for G , querying it in Line 5 and resolving all remaining dependencies in Line 6 clearly solves the graph orientation problem. Note that $V \setminus Q$ is an independent set in G , and thus the nodes in $V \setminus Q$ can only be made mandatory by the results of the queries to Q . Hence, it is known after Line 5 which nodes in $V \setminus Q$ are mandatory, and they can be queried in Line 6 in arbitrary order (or in parallel).

We continue by showing the competitive ratio of $\max\{\frac{1}{d}, \alpha + (2 - \alpha) \cdot d\}$. Algebraic transformations show that the optimal choice for the threshold is $d(\alpha) = 2/(\alpha + \sqrt{8 - \alpha(4 - \alpha)})$. The desired competitive ratio for THRESHOLD with $d = d(\alpha)$ follows.

The algorithm queries set Q and all other vertices only if they are mandatory, hence

$$\mathbb{E}[ALG] = |Q| + \sum_{v \in V \setminus Q} p_v = |M| + |V_1| + |VC'| + \sum_{v \in V_0} p_v + \sum_{v \in V_{1/2} \setminus VC'} p_v. \tag{2}$$

The expected optimal cost can be lower bounded by partitioning and Lemma 2.9:

$$\mathbb{E}[\text{OPT}] \geq \mathbb{E}[\text{OPT}_M] + \mathbb{E}[\text{OPT}_{V_1 \cup V_0}] + \mathbb{E}[\text{OPT}_{V_{1/2}}]. \tag{3}$$

In the remainder we compare $\mathbb{E}[ALG]$ with $\mathbb{E}[\text{OPT}]$ component-wise.

We can lower bound $\mathbb{E}[\text{OPT}_M]$ by $\sum_{v \in M} p_v$ using Equation (1). By definition of M , it holds that $\mathbb{E}[\text{OPT}_M] \geq \sum_{v \in M} p_v \geq d \cdot |M|$. Thus,

$$|M| \leq \frac{1}{d} \cdot \mathbb{E}[\text{OPT}_M]. \tag{4}$$

Next, we compare $|V_1| + \sum_{v \in V_0} p_v$ with $\mathbb{E}[\text{OPT}_{V_1 \cup V_0}]$. For this purpose, let $G[V_1 \cup V_0]$ be the subgraph of G induced by $V_1 \cup V_0$, and let $G'[V_1 \cup V_0]$ be the bipartite graph that is created by removing all edges between elements of V_1 from $G[V_1 \cup V_0]$. It follows from similar arguments as in [47, Theorem 2] that V_1 is a minimum vertex cover of $G'[V_1 \cup V_0]$. This allows us to apply the famous König-Egerváry theorem [52]. By the latter there is a matching h mapping each $v \in V_1$ to a distinct $h(v) \in V_0$ with $\{v, h(v)\} \in E$. Denoting $S = \{h(v) \mid v \in V_1\}$, we can infer $\mathbb{E}[\text{OPT}_{V_1 \cup V_0}] \geq \mathbb{E}[\text{OPT}_{V_1 \cup S}] + \mathbb{E}[\text{OPT}_{V_0 \setminus S}]$.

Any feasible solution must query at least one endpoint of all edges of the form $\{v, h(v)\}$. This implies $\mathbb{E}[\text{OPT}_{V_1 \cup S}] \geq |V_1|$. Since additionally $p_{h(v)} \leq d$ for each $h(v) \in S$, we get

$$\sum_{v \in V_1} (1 + p_{h(v)}) \leq (1 + d) \cdot |V_1| \leq (1 + d) \cdot \mathbb{E}[\text{OPT}_{V_1 \cup S}]. \tag{5}$$

By lower bounding $\mathbb{E}[\text{OPT}_{V_0 \setminus S}]$ with $\sum_{v \in V_0 \setminus S} p_v$ and using (5), we get

$$\begin{aligned} |V_1| + \sum_{v \in V_0} p_v &= \sum_{v \in V_1} (1 + p_{h(v)}) + \sum_{v \in V_0 \setminus S} p_v \\ &\leq (1 + d) \cdot \mathbb{E}[\text{OPT}_{V_1 \cup S}] + \mathbb{E}[\text{OPT}_{V_0 \setminus S}] \leq (1 + d) \cdot \mathbb{E}[\text{OPT}_{V_1 \cup V_0}]. \end{aligned} \quad (6)$$

Finally, consider the term $|VC'| + \sum_{v \in V_{1/2} \setminus VC'} p_v$. Let VC^* be a minimum cardinality vertex cover for $G[V_{1/2}]$. Then, it holds $|VC^*| \geq \frac{1}{2} \cdot |V_{1/2}|$. This is, because in the optimal basic feasible solution to the LP relaxation x^* , each vertex in $V_{1/2}$ has a value of $\frac{1}{2}$. A vertex cover with $|VC^*| < \frac{1}{2} \cdot |V_{1/2}|$ would contradict the optimality of x^* . The following part of the analysis crucially relies on $|VC^*| \geq \frac{1}{2} \cdot |V_{1/2}|$, which is the reason why THRESHOLD executes the LP relaxation-based preprocessing before applying the α -approximation.

Now, the expected cost of the algorithm for the subgraph $G[V_{1/2}]$ is $|VC'| + \sum_{v \in I'} p_v \leq |VC'| + d \cdot |I'|$ with $I' = V_{1/2} \setminus VC'$. Since $|VC'| \geq |VC^*| \geq \frac{1}{2} \cdot |V_{1/2}|$, there is a tradeoff between the quality of $|VC'|$ and the additional cost of $d \cdot |I'|$. If $|VC'|$ is close to $\frac{1}{2} \cdot |V_{1/2}|$, then it is close to $|VC^*|$ but, on the other hand, $|I'|$ then is close to $\frac{1}{2} \cdot |V_{1/2}|$, which means that the additional cost $d \cdot |I'|$ is high. Vice versa, if the cost for $|VC'|$ is high because it is larger than $\frac{1}{2} \cdot |V_{1/2}|$, then $|I'|$ is close to zero and the additional cost $d \cdot |I'|$ is low. We exploit this tradeoff and upper bound $\frac{|VC'| + d \cdot |I'|}{|VC^*|}$ in terms of the approximation factor α of the vertex cover approximation. Assume that the approximation factor α is tight, i.e., $|VC'| = \alpha \cdot |VC^*|$. Since $d \leq 1$, this is the worst case for the ratio $\frac{|VC'| + d \cdot |I'|}{|VC^*|}$. (In other words, if the approximation factor was not tight, we could replace α by the approximation factor that is actually achieved and carry out the following calculations with that smaller value of α instead, yielding an even better bound.) Using $|VC'| = \alpha \cdot |VC^*|$ and $|VC^*| \geq \frac{1}{2} \cdot |V_{1/2}|$, we can derive

$$|I'| = |V_{1/2}| - |VC'| = |V_{1/2}| - \alpha \cdot |VC^*| \leq (2 - \alpha) \cdot |VC^*|.$$

For the cost of the algorithm for subgraph $G[V_{1/2}]$ we get

$$|VC'| + d \cdot |I'| \leq \alpha \cdot |VC^*| + d \cdot (2 - \alpha) \cdot |VC^*| = (\alpha + (2 - \alpha) \cdot d) \cdot |VC^*|.$$

Since $|VC^*| \leq \mathbb{E}[\text{OPT}_{V_{1/2}}]$, we get

$$|VC'| + d \cdot |I'| \leq (\alpha + (2 - \alpha) \cdot d) \cdot \mathbb{E}[\text{OPT}_{V_{1/2}}]. \quad (7)$$

Combining Equations (2), (4), (6) and (7), we can upper bound the cost of the algorithm:

$$\begin{aligned} \mathbb{E}[ALG] &= |M| + |V_1| + \sum_{v \in V_0} p_v + |VC'| + \sum_{v \in V_{1/2} \setminus VC'} p_v \\ &\leq \frac{1}{d} \cdot \mathbb{E}[\text{OPT}_M] + (1 + d) \cdot \mathbb{E}[\text{OPT}_{V_1 \cup V_0}] + (\alpha + (2 - \alpha) \cdot d) \cdot \mathbb{E}[\text{OPT}_{V_{1/2}}] \\ &\leq \max \left\{ \frac{1}{d}, (1 + d), (\alpha + (2 - \alpha) \cdot d) \right\} \cdot \mathbb{E}[\text{OPT}], \end{aligned}$$

where the last inequality follows from the lower bound on OPT in (3). Observe that for any $d \in [0, 1]$ and $\alpha \in [1, 2]$, it holds that $(\alpha + (2 - \alpha) \cdot d) \geq (1 + d)$. We conclude with $\mathbb{E}[ALG] \leq \max \left\{ \frac{1}{d}, (\alpha + (2 - \alpha) \cdot d) \right\} \cdot \mathbb{E}[\text{OPT}]$, which implies the theorem. \blacktriangleleft

In the full version of the paper, we show that the analysis of THRESHOLD is tight. To benefit from a better approximation factor than $\alpha = 2$ for solving the minimum vertex cover problem, we would need to know in advance the specialized graph class on which we want to solve this subproblem. In some cases, we can benefit from optimal or approximation algorithms, e.g., using THRESHOLD with the PTAS for planar graphs [10] allows us to achieve a competitive ratio of at most $1.618 + \epsilon$, for any $\epsilon > 0$, if the input graph is planar.

3.2 Orienting Hypergraphs

► **Theorem 3.2.** *Given an α -approximation with $1 \leq \alpha \leq 2$ for the vertex cover problem (on the induced subgraph $\bar{G}[V_{1/2}]$ of the vertex cover instance given by Definition 2.5, cf. Line 4), a modified version of the THRESHOLD algorithm solves the hypergraph orientation problem with arbitrary query costs with competitive ratio $R = \frac{1}{2} \left(\alpha + \sqrt{\alpha^2 + 4(2 - \alpha)(1 + \alpha\epsilon + (2 - \alpha)\epsilon^2)} + (4 - 2\alpha)\epsilon \right)$ with probability at least $1 - \delta$. Its running time is upper bounded by the complexity of the sampling procedure and the vertex cover black box procedure.*

Proof. The modified algorithm works with the vertex cover instance \bar{G} instead of the given hypergraph H , following Definition 2.5. In Line 1, we use $d(\alpha) = 1/R + \epsilon$. In Line 6, we iteratively query mandatory vertices until the instance is solved. In addition, we use a random estimation Y_v of p_v , instead of the precise probability, using the procedure described in Lemma 2.2 with parameters ϵ and δ' such that $1 - \delta = (1 - \delta')^n$. As a result, with probability at least $1 - \delta$ we have that for every vertex v , the estimation Y_v has absolute error at most ϵ . In case of this event we obtain the following bound on the cost (which is optimized for the chosen value of d), namely $\mathbb{E}[\text{ALG}] \leq \max\{\frac{1}{d-\epsilon}, (1+d+\epsilon), (\alpha+(2-\alpha)(d+\epsilon))\} \cdot \mathbb{E}[\text{OPT}]$. ◀

Sorting a set of elements is equivalent to determining, for each pair of elements, which of the two has smaller weight. Hence, the problem of sorting multiple sets of elements with uncertain weights is a special case of the graph orientation problem: For each set to be sorted, the edge set of a complete graph on its elements is added to a graph, and the resulting instance of the graph orientation problem is then equivalent to the given instance of the sorting problem. We can show the following theorem.

► **Theorem 3.3.** *For the special cases of orienting $\mathcal{O}(\log |V|)$ hyperedges and sorting $\mathcal{O}(1)$ sets, THRESHOLD can be applied with $\alpha = 1$ in polynomial running time.*

4 Vertex Cover-Based Algorithms: Improved Results for Special Cases

Consider an arbitrary vertex cover-based algorithm ALG. It queries a vertex cover VC in the first stage, and continues with elements of $V \setminus VC$ if they are mandatory. Thus,

$$\begin{aligned} \mathbb{E}[\text{ALG}] &= c(VC) + \sum_{v \in V \setminus VC} p_v \cdot c_v = \sum_{v \in VC} (p_v \cdot c_v + (1 - p_v) \cdot c_v) + \sum_{v \in V \setminus VC} p_v \cdot c_v \\ &= \sum_{v \in V} p_v \cdot c_v + \sum_{v \in VC} (1 - p_v) \cdot c_v. \end{aligned}$$

Since the first term is independent of VC , ALG is the best possible vertex cover-based algorithm if it minimizes $\sum_{v \in VC} (1 - p_v) \cdot c_v$. We refer to this algorithm as BESTVC.

To implement BESTVC, we need the exact value p_v , for all $v \in V$, and an optimal algorithm for computing a weighted vertex cover. As mentioned in Section 2, the first problem is #P-hard in hypergraphs, but it can be solved exactly in polynomial time for graphs. The weighted vertex cover problem can be solved optimally in polynomial time for bipartite graphs.

In general, BESTVC has competitive ratio at least 1.5 (Theorem 2.6). However, we show in the following that it is $4/3$ -competitive for two special cases. It remains open whether BESTVC still outperforms THRESHOLD if the vertex cover is only approximated with a factor $\alpha > 1$.

4.1 A Best Possible Algorithm for Orienting Bipartite Graphs

► **Theorem 4.1.** *BESTVC is $\frac{4}{3}$ -competitive for the bipartite graph orientation problem.*

Proof. In the full version of the paper, we show that BESTVC is $\frac{4}{3}$ -competitive for the problem of orienting stars if both vertex cover-based algorithms (either querying the leaves or the center first) have the same expected cost. In this proof, we divide the instance into subproblems that fulfill these requirements, and use the result for stars to infer $\frac{4}{3}$ -competitiveness for bipartite graphs.

Let VC be a minimum-weight vertex cover (with weights $c_v \cdot (1 - p_v)$) as computed by BESTVC in the first phase. By the König-Egerváry theorem (e.g., [52]), there is a function $\pi : E \rightarrow \mathbb{R}$ with $\sum_{\{u,v\} \in E} \pi(u,v) \leq c_v \cdot (1 - p_v)$ for each $v \in V$. By duality theory, the constraint is tight for each $v \in VC$, and $\pi(u,v) = 0$ holds if both u and v are in VC . Thus, we can interpret π as a function that distributes the weight of each $v \in VC$ to its neighbors outside of VC .

For each $v \in VC$ and $u \in V \setminus VC$, let $\lambda_{u,v} := \frac{\pi(u,v)}{(1-p_u) \cdot c_u}$ denote the fraction of the weight of u that is used by π to cover the weight of v . Moreover, for $u \in V \setminus VC$, let $\tau_u := 1 - \sum_{\{u,v\} \in E} \lambda_{u,v}$ be the fraction of the weight of u that is not used by π to cover the weight of any $v \in VC$. Then, we can write the expected cost of BESTVC as follows:

$$\mathbb{E}[\text{BESTVC}] = \sum_{v \in VC} \left(c_v + \sum_{u \in V \setminus VC} p_u \cdot \lambda_{u,v} \cdot c_u \right) + \sum_{u \in V \setminus VC} p_u \cdot \tau_u \cdot c_u. \quad (8)$$

Using Observation 2.10, we compare $\mathbb{E}[\text{BESTVC}]$ with the expected optimum $\mathbb{E}[\text{OPT}']$ for an instance $G' = (V', E', c')$ that is created by splitting vertices. We modify the mandatory distribution as in Section 2, which implies $\mathbb{E}[\text{OPT}] = \mathbb{E}[\text{OPT}']$. We add the following copies:

1. For each $v \in VC$, we add a copy v' of v to V' with $c'_{v'} = c_v$.
2. For all $u \in V \setminus VC$ and $v \in VC$ with $\lambda_{u,v} > 0$, we add a copy u'_v of u to V' with $c'_{u'_v} = \lambda_{u,v} \cdot c_u$.
3. For each $u \in V \setminus VC$ with $\tau_u > 0$, we add a copy u' of u to V' with $c'_{u'} = \tau_u \cdot c_u$.

Let p'_v denote the probability of v being mandatory for instance G' . By definition of the vertex split operation, we have $p_v = p'_u$ for each $v \in V$ and each copy $u \in V'$ of v . Further, for each $v \in VC$, define $H'_v = \{u'_v \mid u \in V \text{ with } \lambda_{u,v} > 0\}$. By definition of H'_v and G' , all H'_v are pairwise disjoint. Let $\mathcal{H}' = \bigcup_{v \in VC} (H'_v \cup \{v'\})$; then we can express $\mathbb{E}[\text{BESTVC}]$ as follows:

$$\begin{aligned} \mathbb{E}[\text{BESTVC}] &= \sum_{v \in VC} \left(c_v + \sum_{u \in V \setminus VC} p_u \cdot \lambda_{u,v} \cdot c_u \right) + \sum_{u \in V \setminus VC} p_u \cdot \tau_u \cdot c_u \\ &= \sum_{v \in VC} \left(c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u \right) + \sum_{u \in V' \setminus \mathcal{H}'} p'_u \cdot c'_u. \end{aligned}$$

Similarly, we can lower bound $\mathbb{E}[\text{OPT}']$ using Lemma 2.9:

$$\mathbb{E}[\text{OPT}'] \geq \sum_{v \in VC} \mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}] + \sum_{u \in V' \setminus \mathcal{H}'} \mathbb{E}[\text{OPT}'_{\{u\}}] \geq \sum_{v \in VC} \mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}] + \sum_{u \in V' \setminus \mathcal{H}'} p'_u \cdot c'_u. \quad (9)$$

Since the term $\sum_{u \in V' \setminus \mathcal{H}'} p'_u \cdot c'_u$ shows up in both inequalities, it remains, for each $v \in VC$, to bound $c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u$ in terms of $\mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}]$. By definition of H'_v , we have $(1 - p'_{v'}) \cdot c'_{v'} = \sum_{u \in H'_v} (1 - p'_u) \cdot c'_u$, which implies

$$c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u = p'_{v'} \cdot c'_{v'} + \sum_{u \in H'_v} c'_u. \quad (10)$$

The value $\mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}]$ corresponds to the expected optimum for the subproblem which considers the subgraph induced by $H'_v \cup \{v'\}$ (which is a star), uses p'_u as the mandatory probability for each $u \in H'_v \cup \{v'\}$, and uses c'_u as the query cost of each $u \in H'_v \cup \{v'\}$. For this subproblem, $c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u$ corresponds to the expected cost of the vertex cover-based algorithm that queries vertex cover $\{v'\}$ in the first stage. Furthermore, $p'_{v'} \cdot c'_{v'} + \sum_{u \in H'_v} c'_u$ corresponds to the expected cost of the vertex-cover based algorithm that queries vertex cover H'_v in the first stage. In summary, we have a star orientation subproblem for which both vertex cover-based algorithms (querying the leaves or the center first) have the same expected cost (cf. Equation (10)). As a major technical step, we show that BESTVC is $\frac{4}{3}$ -competitive for such subproblems, which implies

$$c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u \leq \frac{4}{3} \cdot \mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}].$$

A corresponding lemma is proven in the full version. We remark that the lemma requires $p'_{v'}$ to be independent of each p'_u with $u \in H'_v$; otherwise the subproblem does not correspond to the star orientation problem. As the input graph is bipartite, such independence follows by definition.

Using this inequality and Equation (9), we conclude that BESTVC is $4/3$ -competitive.

$$\begin{aligned} \mathbb{E}[\text{BESTVC}] &= \sum_{v \in VC} \left(c'_{v'} + \sum_{u \in H'_v} p'_u \cdot c'_u \right) + \sum_{u \in V' \setminus \mathcal{H}'} p'_u \cdot c'_u \\ &\leq \frac{4}{3} \cdot \sum_{v \in VC} \mathbb{E}[\text{OPT}'_{H'_v \cup \{v'\}}] + \sum_{u \in V' \setminus \mathcal{H}'} p'_u \cdot c'_u \leq \frac{4}{3} \cdot \mathbb{E}[\text{OPT}'] = \frac{4}{3} \cdot \mathbb{E}[\text{OPT}] \blacktriangleleft \end{aligned}$$

4.2 An Almost Best Possible Algorithm for Orienting a Hyperedge

► **Theorem 4.2.** *BESTVC has a competitive ratio at most $\min\{\frac{4}{3}, \frac{n+1}{n}\}$ for the hypergraph orientation problem on a single hyperedge with $n \geq 2$ vertices and uniform query costs. For a hyperedge with only two vertices, the algorithm is 1.207-competitive.*

Our analysis improves upon a $(n+1)/n$ -competitive algorithm by Chaplick et al. [16] in case that the hyperedge has two or three vertices. Moreover, we show that this is near-optimal: It is not hard to show a matching lower bound for two vertices and, due to Theorem 2.3 and the theorem below, this is the best possible for three vertices, and in general the difference between the upper and lower bounds is less than 4%.

► **Theorem 4.3.** *Any algorithm for orienting a single hyperedge with $n+1 \geq 2$ vertices has competitive ratio at least $n^2/(n^2 - n + 1)$, even for uniform query costs.*

Note that Theorem 4.2 is in contrast to the problem of orienting a hyperedge with arbitrary query costs: In this setting, [16] showed that the algorithm is 1.5-competitive, which matches the corresponding lower bound for vertex cover-based algorithms of Theorem 2.6.

5 Conclusion

In this paper, we present algorithms for the (hyper)graph orientation problem under stochastic explorable uncertainty. It remains open to determine the competitive ratio of BESTVC for the general (hyper)graph orientation problem, and to investigate how the algorithm behaves if it has to rely on an α -approximation to solve the vertex cover subproblem. In

this context, one can consider the resulting algorithm as a standalone algorithm, or as a subroutine for THRESHOLD. Our analysis suggests that, to achieve a competitive ratio better than 1.5, algorithms have to employ more adaptivity; exploiting this possibility remains an open problem. Finally, it would be interesting to characterize the vertex cover instances arising in our THRESHOLD algorithm. In addition to the relevance from a combinatorial point of view, such a characterization may allow an improved α -approximation algorithm for those instances.

References


- 1 Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. *Math. Oper. Res.*, 41(3):1022–1038, 2016.
- 2 Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *WAOA 2020: 18th International Workshop on Approximation and Online Algorithms*, 2021.
- 3 Luciana Arantes, Evripidis Bampis, Alexander V. Kononov, Manthos Letsios, Giorgio Lucarelli, and Pierre Sens. Scheduling under uncertainty: A query-based approach. In *IJCAI 2018: 27th International Joint Conference on Artificial Intelligence*, pages 4646–4652, 2018. doi: 10.24963/ijcai.2018/646.
- 4 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *ESA 2021: 29th Annual European Symposium on Algorithms*, volume 204 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 5 Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. *ACM Trans. Economics and Comput.*, 7(3):16:1–16:19, 2019.
- 6 Evripidis Bampis, Christoph Dürr, Thomas Erlebach, Murilo S. de Lima, Nicole Megow, and Jens Schlöter. Orienting (hyper)graphs under explorable stochastic uncertainty. *CoRR*, abs/2107.00572, 2021. URL: <https://arxiv.org/abs/2107.00572>.
- 7 Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.
- 8 Nikhil Bansal and Viswanath Nagarajan. On the adaptivity gap of stochastic orienteering. *Math. Program.*, 154(1-2):145–172, 2015.
- 9 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. In memoriam: Shimon Even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004. doi:10.1145/1041680.1041683.
- 10 Reuven Bar-Yehuda and Shimon Even. On approximating a vertex cover for planar graphs. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC 1982: 14th Annual ACM Symposium on Theory of Computing*, pages 303–309. ACM, 1982. doi:10.1145/800070.802205.
- 11 Paul Beame, Sarel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94 of *LIPIcs*, pages 38:1–38:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPIcs.ITCS.2018.38.
- 12 Soheil Behnezhad, Alireza Farhadi, MohammadTaghi Hajiaghayi, and Nima Reyhani. Stochastic matching with few queries: New algorithms and tools. In *SODA 2019: 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2855–2874. SIAM, 2019.
- 13 Avrim Blum, John P. Dickerson, Nika Haghtalab, Ariel D. Procaccia, Tuomas Sandholm, and Ankit Sharma. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. *Oper. Res.*, 68(1):16–34, 2020.

- 14 Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.
- 15 Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.
- 16 Steven Chaplick, Magnúús M. Halldórsson, Murilo S. de Lima, and Tigran Tonoyan. Query minimization under stochastic uncertainty. In Y. Kohayakawa and F. K. Miyazawa, editors, *LATIN 2020: 14th Latin American Theoretical Informatics Symposium*, volume 12118 of *Lecture Notes in Computer Science*, pages 181–193. Springer Berlin Heidelberg, 2020.
- 17 Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP 2009: 36th International Colloquium on Automata, Languages and Programming, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 266–278. Springer, 2009. doi:10.1007/978-3-642-02927-1_23.
- 18 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In Shuchi Chawla, editor, *SODA 2020: Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 2916–2935. SIAM, 2020. doi:10.1137/1.9781611975994.177.
- 19 Miroslav Chlebik and Janka Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM Journal on Discrete Mathematics*, 21(1):158–169, 2007.
- 20 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.
- 21 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020.
- 22 Thomas Erlebach and Michael Hoffmann. Minimum spanning tree verification under uncertainty. In D. Kratsch and I. Todinca, editors, *WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-319-12340-0_14.
- 23 Thomas Erlebach, Michael Hoffmann, and Murilo S. de Lima. Round-competitive algorithms for uncertainty problems with parallel queries. In Markus Bläser and Benjamin Monmege, editors, *STACS 2021: 38th International Symposium on Theoretical Aspects of Computer Science*, volume 187 of *LIPIcs*, pages 27:1–27:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.27.
- 24 Thomas Erlebach, Michael Hoffmann, Murilo S. de Lima, Nicole Megow, and Jens Schlöter. Untrusted predictions improve trustable query policies. *CoRR*, abs/2011.07385, 2020. URL: <https://arxiv.org/abs/2011.07385>.
- 25 Thomas Erlebach, Michael Hoffmann, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS’08: 25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *LIPIcs*, pages 277–288. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.
- 26 Tomás Feder, Rajeev Motwani, Liadan O’Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.
- 27 Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.
- 28 Jacob Focke, Nicole Megow, and Julie Meißner. Minimum spanning tree under explorable uncertainty in theory and experiments. *ACM J. Exp. Algorithmics*, 25:1–20, 2020. doi:10.1145/3422371.
- 29 András Frank. *Connections in Combinatorial Optimization*. Oxford University Press, USA, 2011.


- 30 John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed Bandit Allocation Indices*. Wiley, 2nd edition, 2011.
- 31 Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Computers & Operations Research*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.
- 32 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.
- 33 Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. The Markovian price of information. In *IPCO 2019: 20th International Conference on Integer Programming and Combinatorial Optimization*, volume 11480 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 2019.
- 34 Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Running errands in time: Approximation algorithms for stochastic orienteering. *Math. Oper. Res.*, 40(1):56–79, 2015.
- 35 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *IPCO 2013: 16th International Conference on Integer Programming and Combinatorial Optimization*, volume 7801 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2013.
- 36 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and adaptivity gaps for stochastic probing. In *SODA 2016: 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1731–1747. SIAM, 2016.
- 37 Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. The update complexity of selection and related problems. *Theory of Computing Systems*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.
- 38 Magnús M. Halldórsson and Murilo Santos de Lima. Query-competitive sorting with uncertainty. *Theor. Comput. Sci.*, 867:50–67, 2021. doi:10.1016/j.tcs.2021.03.021.
- 39 Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.
- 40 Simon Kahan. A model for data in motion. In *STOC’91: 23rd Annual ACM Symposium on Theory of Computing*, pages 265–277, 1991. doi:10.1145/103418.103449.
- 41 Sanjeev Khanna and Wang-Chiew Tan. On computing functions with uncertainty. In *PODS’01: 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 171–182, 2001. doi:10.1145/375551.375577.
- 42 Retsef Levi, Thomas L. Magnanti, and Yaron Shaposhnik. Scheduling with testing. *Manag. Sci.*, 65(2):776–793, 2019.
- 43 Will Ma. Improvements and generalizations of stochastic knapsack and Markovian bandits approximation algorithms. *Math. Oper. Res.*, 43(3):789–812, 2018.
- 44 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In Moses Charikar, editor, *SODA 2010: 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 608–615. SIAM, 2010. doi:10.1137/1.9781611973075.51.
- 45 Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.
- 46 Arturo I. Merino and José A. Soto. The minimum cost query problem on matroids with uncertainty areas. In *ICALP 2019: 46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *LIPIcs*, pages 83:1–83:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 47 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.
- 48 Noam Nisan. The demand query model for bipartite matching. In *SODA 2021: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 592–599. SIAM, 2021. doi:10.1137/1.9781611976465.36.

- 49 Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB 2000: 26th International Conference on Very Large Data Bases*, pages 144–155, 2000. URL: <http://ilpubs.stanford.edu:8090/437/>.
- 50 Herbert E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939. URL: <http://www.jstor.org/stable/2303897>.
- 51 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.39.
- 52 Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.
- 53 Sahil Singla. The price of information in combinatorial optimization. In *SODA 2018: 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2523–2532. SIAM, 2018. doi:10.1137/1.9781611975031.161.
- 54 William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.
- 55 Martin Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–54, 1979.
- 56 Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980. doi:10.1137/0138030.

k -Distinct Branchings Admits a Polynomial Kernel

Jørgen Bang-Jensen 

University of Southern Denmark, Odense, Denmark

Kristine Vitting Klinkby 

University of Bergen, Bergen, Norway

University of Southern Denmark, Odense, Denmark

Saket Saurabh 

University of Bergen, Bergen, Norway

The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

Unlike the problem of deciding whether a digraph $D = (V, A)$ has ℓ in-branchings (or ℓ out-branchings) is polynomial time solvable, the problem of deciding whether a digraph $D = (V, A)$ has an in-branching B^- and an out-branching B^+ which are arc-disjoint is NP-complete. Motivated by this, a natural optimization question that has been studied in the realm of Parameterized Complexity is called ROOTED k -DISTINCT BRANCHINGS. In this problem, a digraph $D = (V, A)$ with two prescribed vertices s, t are given as input and the question is whether D has an in-branching rooted at t and an out-branching rooted at s such that they differ on at least k arcs. Bang-Jensen et al. [Algorithmica, 2016] showed that the problem is fixed parameter tractable (FPT) on strongly connected digraphs. Gutin et al. [ICALP, 2017; JCSS, 2018] completely resolved this problem by designing an algorithm with running time $2^{\mathcal{O}(k^2 \log^2 k)} n^{\mathcal{O}(1)}$. Here, n denotes the number of vertices of the input digraph. In this paper, answering an open question of Gutin et al., we design a polynomial kernel for ROOTED k -DISTINCT BRANCHINGS. In particular, we obtain the following: Given an instance (D, k, s, t) of ROOTED k -DISTINCT BRANCHINGS, in polynomial time we obtain an equivalent instance (D', k', s, t) of ROOTED k -DISTINCT BRANCHINGS such that $|V(D')| \leq \mathcal{O}(k^2)$ and the treewidth of the underlying undirected graph is at most $\mathcal{O}(k)$. This result immediately yields an FPT algorithm with running time $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$; improving upon the previous running time of Gutin et al. For our algorithms, we prove a structural result about paths avoiding many arcs in a given in-branching or out-branching. This result might turn out to be useful for getting other results for problems concerning in-and out-branchings.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Digraphs, Polynomial Kernel, In-branching, Out-Branching

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.11

1 Introduction

Let $D = (V, A)$ be a digraph and r be a vertex of D . An *out-branching* (respectively, *in-branching*) in D is a connected spanning subdigraph B_r^+ (respectively, B_r^-) of D in which each vertex $v \neq r$ has precisely one entering (respectively, leaving) arc and r has no entering (respectively, leaving) arc. The vertex r is called the *root* of B_r^+ (respectively, B_r^-). The study of finding a spanning tree in an undirected graph or an out-branching in a digraph satisfying specific properties, such as having at least k leaves, or having at least k internal vertices [1, 4, 7, 10, 11, 13, 14, 18, 19, 22] has been at the forefront of research in parameterized algorithms. This paper aims to study a problem of finding an in-branching and an out-branching, in the given digraph, whose arc sets is disjoint on at least k arcs, in the realm of Kernelization Complexity [21] and Parameterized Complexity [12, 15, 17, 24].

A parameterized problem Π is said to admit a *kernel* if there is a polynomial-time algorithm, called a *kernelization algorithm*, that reduces the input instance of Π down to an equivalent instance of Π whose size is bounded by a function $f(k)$ of k . (Here, two instances



© Jørgen Bang-Jensen, Kristine Vitting Klinkby, and Saket Saurabh;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 11; pp. 11:1–11:15

Leibniz International Proceedings in Informatics



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

are equivalent if both of them are either yes-instances or no-instances.) Such an algorithm is called an $f(k)$ -kernel for Π . If $f(k)$ is a polynomial function of k , we say that the kernel is a *polynomial kernel*. See [12, 15, 17, 21, 24] for more details.

In studying problems around finding edge-disjoint spanning trees, arc-disjoint in-branchings, or arc-disjoint in-branchings, one of the most important results is due to Edmonds. This classical result states that given a digraph D and a positive integer ℓ , we can test whether D contains ℓ arc-disjoint out-branchings in polynomial time [16]. In contrast to this, Thomassen proved that the problem of deciding whether a digraph contains an out-branching B_s^+ and an in-branching B_t^- which are arc-disjoint is **NP**-complete, even if $s = t$ (the proof is published in [2]). The problem remains **NP**-complete even for 2-regular digraphs [5] but it is polynomial time solvable on tournaments [2], locally semicomplete digraphs [3] and acyclic digraphs [6, 9]. In fact, even deciding whether a digraph D contains an out-branching which is arc-disjoint from some spanning tree in the underlying undirected graph remains **NP**-complete [8].

In this paper, we consider the following parameterized version of the arc-disjoint in- and out-branching problem by using a parameter k to measure how distinct a given pair B_s^+, B_t^- are, where the measure is in terms of the number of arcs that belongs to B_s^+ but not to B_t^- . In particular, we study the following problem.

ROOTED k -DISTINCT BRANCHINGS (R- k -DB)

Parameter: k

Input: A digraph $D = (V, A)$, two fixed vertices $s, t \in V$ and an integer k .

Question: Does there exist an out-branching B_s^+ and an in-branching B_t^- such that $|A(B_s^+) \setminus A(B_t^-)| \geq k$?

Observe that the problem is **NP**-complete since it contains the arc-disjoint in- and out-branching question as to the particular case when k is the number of vertices minus one.

Context of our Study. The problem R- k -DB has a rich history in the realm of Parameterized Complexity. Bang-Jensen and Yeo [7] asked whether the problem would be **FPT** when $s = t$. Answering this question in the affirmative, Bang-Jensen et al. [4] showed that the problem is **FPT** when the input is a strongly connected digraph and asked *whether the problem is FPT on general digraphs*. This was confirmed in affirmative by Gutin et al. [22], who showed that the problem is solvable in time $2^{\mathcal{O}(k^2 \log^2 k)} n^{\mathcal{O}(1)}$ time. A natural follow up question that they ask is: *Does R- k -DB admits a polynomial sized kernel?* This open question is the starting point of our work.

Our Results and Methods. We design a polynomial kernel for R- k -DB.

► **Theorem 1.1.** *R- k -DB admits a polynomial kernel with $\mathcal{O}(k^2)$ vertices.*

A key ingredient in the work of Gutin et al. [22] is out-branchings with many leaves. A vertex v is a leaf in the out-branching B_s^+ if no arc is leaving v in B_s^+ . If the input (D, s, t, k) to the R- k -DB problem is such that D has an out-branching B_s^+ with at least $k + 1$ leaves, then (D, s, t, k) is a “yes”-instance since every in-branching B_t^- will have the property that no arc of B_t^- which leaves a vertex in L will be contained in B_s^+ , where L is the set of leaves of B_s^+ . It was shown in [13, 23] that the problem (parameterized by p) of deciding the existence of an out-branching with at least p leaves is **FPT**. Furthermore, if the input is strongly connected and has no out-branching B_s^+ with at least $k + 1$ leaves, then it has pathwidth

$\mathcal{O}(k \log k)$ and now a result from [4] implies that the ROOTED k -DISTINCT BRANCHINGS problem is FPT for strongly connected digraphs. When the input is not strongly connected, the case is considerably more complicated to handle when following the approach used in [22]. Indeed, the result of Gutin et al. [22] could be viewed as an algorithm that obtains an equivalent instance with treewidth of the underlying graph bounded by $k^{\mathcal{O}(1)}$. Our approach is very different, as we bound the size rather than a structural property.

We use a fundamentally different approach, based on a structural analysis of paths avoiding many arcs of a fixed out-branching, to prove an $\mathcal{O}(k^2)$ -vertex kernel for the R- k -DB problem. In the whole analysis, we work with a fixed out-branching and see how an in-branching can be built that avoids as many arcs of the given out-branching. In a step-by-step procedure, we either obtain reduction rules to reduce our input or at the end, we do get an in-branching and an out-branching that avoids k -arcs of each other. Finally, we argue that the instance on which none of the reduction rules can be applied has at most $\mathcal{O}(k^2)$ vertices.

We further look into our kernel and try to bound the treewidth of the underlying undirected graph. In particular, given an instance (D, k, s, t) of R- k -DB, in polynomial time we obtain an equivalent instance (D', k', s, t) of R- k -DB such that $|V(D')| \leq \mathcal{O}(k^2)$ and the treewidth of the underlying undirected graph is at most $\mathcal{O}(k)$. This result yields the FPT algorithm by a standard dynamic programming approach over graphs of bounded treewidth.

► **Theorem 1.2** (\star).¹ *R- k -DB admits an algorithm with running time $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$.*

The running time obtained in Theorem 1.2 improves upon the running time of Gutin et al. [22]. We conclude by saying that the structural result on so-called *substitute paths of an out-branching* might be of independent interest and could find further applications.

2 Notation and Preliminaries

Given a digraph $D = (V, A)$, we also use $V(D)$ and $A(D)$ to denote the vertex set and the arc set of D , respectively. If it is clear from the context we simply use V and A , respectively. Given a digraph $D = (V, A)$ and an arc $a = (u, v) \in A$ we call u the *tail* of a and v the *head* of a . The vertices u and v are the *endpoints* of a . For a set of arcs $A' \subseteq A$ the *set of heads* denoted $\mathcal{H}(A')$ is the set of heads of the arcs in A' . That is, $\mathcal{H}(A') = \{h \mid (t, h) \in A' \text{ for some } h, t \in V\}$. Similarly, we have the *set of tails* denoted $\mathcal{T}(A')$, which is defined as follows: $\mathcal{T}(A') = \{t \mid (t, h) \in A' \text{ for some } h, t \in V\}$. In a digraph D we will denote a path from u to v as $P_{u,v}$. By $P_{u,v}[x, y]$ we will denote the subpath of $P_{u,v}$ which goes from x to y . We will denote the path $P_{u,v}[x, y] - \{y\}$ as $P_{u,v}[x, y[$ and the path $P_{u,v}[x, y] - \{x\}$ as $P_{u,v}[x, y]$. Given a path $P_{u,v}$ and $P_{v,z}$ we will describe the concatenation of the paths $P_{u,v}$ and $P_{v,z}$ as $P_{u,z} = P_{u,v}P_{v,z}$. For a vertex $u \in V(D)$ and an arc $(u, v) \in A(D)$ we will often use the notation $u \in D$ and $(u, v) \in D$ respectively. For a vertex $v \in V(D)$ we denote the out-neighbors of v as $N^+(v)$ and the in-neighbors as $N^-(v)$. An out-branching with a root $r \in V$ is denoted B_r^+ . For a vertex $u \in V(B_r^+)$ we say that v is *parent* of u if $(v, u) \in A(B_r^+)$. To indicate that v is parent of u we use the notation $v = \mathcal{P}_{B_r^+}(u)$. We denote the directed path from u to v in B_r^+ as $B_r^+[u, v]$ and we say that u is *ancestor* to v in B_r^+ if the path $B_r^+[u, v]$ exists. An in-branching with a root $r \in V$ is denoted B_r^- . For a vertex $u \in V(B_r^-)$ we say that v is *parent* of u if $(u, v) \in A(B_r^-)$. To indicate this we use the notation $v = \mathcal{P}_{B_r^-}(u)$. We denote the directed path from u to v in B_r^- as $B_r^-[u, v]$ and we say that u is *ancestor* to v in B_r^- if the path $B_r^-[u, v]$ exists.

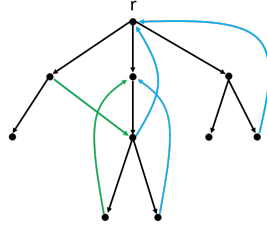
¹ Proofs labeled with \star is not included in this version.

3 Backward Arcs, Crossing Arcs and Substitute-paths

In this section we initiate our structural analysis of substitute paths. Given an out-branching B_r^+ in a digraph $D = (V, A)$, we will work with the arcs in $A(D) \setminus A(B_r^+)$. We therefore divide these arcs into backward and crossing arcs.

► **Definition 1.** *Given an out-branching B_r^+ in a digraph $D = (V, A)$ an arc $(u, v) \in A(D) \setminus A(B_r^+)$ is **backward**, if the path $B_r^+[v, u]$ exists, and **crossing**, if it is not backward.*

In an out-branching B_r^+ the arc $(u, v) \in A$ is backward if and only if v is an ancestor of u . We say that a backward arc (u, v) for B_r^+ **goes over** a vertex p if $p \in B_r^+[v, u]$. It means that a backward arc $(u, v) \in B_r^+$ always goes over u . Figure 1 shows an out-branching B_r^+ together with crossing and backward arcs in B_r^+ .



■ **Figure 1** The figure shows $D = (V, A)$. An out-branching B_r^+ is shown in black, the crossing arcs for B_r^+ are shown in green, and the backwards arcs are shown in blue.

► **Definition 2.** *For an out-branching B_r^+ in a digraph D a backward arc (u, v) is **irrelevant** with respect to B_r^+ if and only if every path from r to u in D contains v . A backward arc which is not irrelevant is **relevant**.*

Definition 2 implies the following.

► **Observation 3.** *An arc a is irrelevant wrt² some out-branching B_r^+ if and only if a is not contained in any out-branching B_r^{*+} , that is, a is irrelevant in any out-branching B_r^{*+} .*

In the rest of this section we consider a fixed out-branching B_r^+ , unless stated otherwise.

► **Lemma 3.1.** *It is possible to find the set of all relevant arcs and the set of all irrelevant arcs for B_r^+ in polynomial time.*

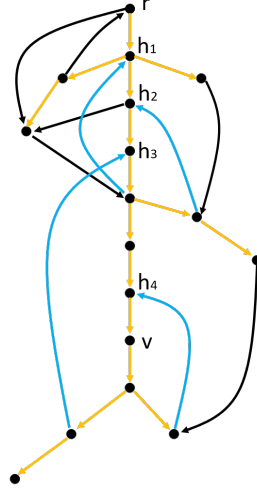
Proof. For any arc $(u, v) \in A(D) \setminus A(B_r^+)$ it can clearly be determined in linear time if the path $B_r^+[v, u]$ exists and, therefore, whether the arc is backward. For every backward arc $(u, v) \in A(D)$ we can also determine in polynomial time if there exists a path $P_{r,u}$ in D which does not contain v , e.g., by determine if a (r, u) -path exists in $D - \{v\}$. ◀

Given the set of relevant arcs wrt B_r^+ in D we can now define the following relation between the relevant arcs and a vertex $u \in V$.

► **Definition 4.** *Let $R(u)$ be the set of those relevant arcs wrt B_r^+ which go over u . The **joint relevant arc set** for u is (recursively) defined as the arc set $J(u) = R(u) \cup \left(\bigcup_{h \in \mathcal{H}(R(u))} J(h) \right)$.*

² We use wrt for an abbreviation of “with respect to”.

Hence for every vertex $u \in B_r^+$ the joint relevant arc set for u is defined recursively as the union of $R(u)$ and the joint relevant arc sets of all the heads of $R(u)$. Figure 2 shows a joint relevant arc set of a vertex v in a out-branching B_r^+ . For notational purpose we let $H(u)$ denote the set $\mathcal{H}(J(u))$.



■ **Figure 2** The Figure shows D . $A(B_r^+)$ is shown in yellow and $J(v)$ is shown in blue.

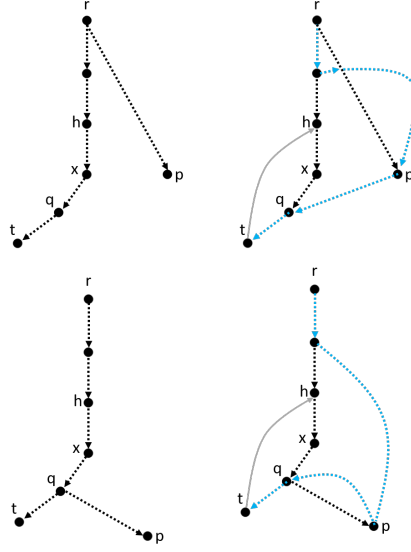
► **Lemma 3.2.** *For every vertex $u \in V$ such that $\mathcal{H}(u) \neq \emptyset$, it holds that for the last vertex x on $B_r^+[r, u]$ which is also in $\mathcal{H}(u)$ we have $\mathcal{H}(x) = \mathcal{H}(u) - \{x\}$.*

In Figure 2 we have $\mathcal{H}(v) = \{h_1, h_2, h_3, h_4\}$. Lemma 3.2 means that $\mathcal{H}(h_4) = \{h_1, h_2, h_3\}$. The following is the proof for Lemma 3.2.

Proof. Let $u \in V$ be a vertex such that $\mathcal{H}(u) \neq \emptyset$ and let $l = |\mathcal{H}(u)|$. Observe that for every $h \in \mathcal{H}(u)$ it holds that $h \in B_r^+[r, u]$. Now name the vertices in $\mathcal{H}(u)$ as h_1, h_2, \dots, h_l such that h_j comes before h_i on $B_r^+[r, u]$ if $j < i$. Let i be an integer in the range $[1, l-1]$. Since $h_i \in \mathcal{H}(u)$ it holds that there exists a relevant arc (t, h_i) which goes over either u or a head $h_p \in \mathcal{H}(u)$ where $i < p \leq l$. Observe that $h_{i+1} \in B_r^+[h_i, u]$ and $h_{i+1} \in B_r^+[h_i, h_j]$ for $j > i$. Hence (t, h_i) goes over h_{i+1} . Therefore, $h_i \in \mathcal{H}(h_{i+1})$. By induction it means that for $i > 0$ we have $\{h_1, h_2, \dots, h_{i-1}\} \subseteq \mathcal{H}(h_i)$. Now observe that no backward arc with the head h_j for $j \geq i$ can go over h_i as $h_j \in B_r^+[h_i, u]$. Hence $\{h_i, h_{i+1}, \dots, h_l\} \cap \mathcal{H}(h_i) = \emptyset$. As $h_i \in \mathcal{H}(u)$ we have that $J(h_i) \subseteq J(u)$. Hence $\mathcal{H}(h_i) \subseteq \mathcal{H}(u)$. As a result $\mathcal{H}(h_i) = \{h_1, h_2, \dots, h_{i-1}\}$. For $i = l$ we therefore have that $\mathcal{H}(h_l) = \{h_1, h_2, \dots, h_{l-1}\} = \mathcal{H}(u) - \{h_l\}$. ◀

► **Lemma 3.3.** *Let C be the set of crossing arcs wrt. to B_r^+ . Then for every $v \in V$ we have $H(v) \subseteq H(u)$ for some arc $(w, u) \in C$, where it is possible that $v = u$.*

Proof. Assume for contradiction that there exists a vertex $x \in V$ such that $H(x) \not\subseteq H(u)$ for every arc $(v, u) \in C$ where we set $H(u) = \emptyset$ if $C = \emptyset$. Among all such vertices choose x such that $|H(x)|$ is maximized. Recall that for every $h \in H(x)$ we have $h \in B_r^+[r, x]$. Now let h be the last vertex on $B_r^+[r, x]$ which is also in $H(x)$. Observe $h \in H(x)$ and there are no vertices in $B_r^+[h, x] \cap H(x)$. Therefore, there must exist an arc $(t, h) \in R(x)$ which goes over x , and since it is relevant there is a path $P_{r,t}$ in D which does not contain h . As a consequence, there is an arc $(p, q) \in P_{r,t}$ which is either a backward arc or a crossing arc such that its head is in $P_{r,t} \cap B_r^+[h, t]$. Figure 3 shows two possible situations where the arc (p, q) is contained in $P_{r,t}$.



■ **Figure 3** In the top left of the Figure a possible section of B_r^+ is shown in black. In the top right, the path $P_{r,t}$ is shown in blue and the arc (t, h) is shown in gray. The bottom left figure shows another possible section of B_r^+ and the bottom right figure shows the path $P_{r,t}$ in blue and the arc (t, h) in gray.

Observe that since $q \in B_r^+[h, t]$ the arc (t, h) goes over q , implying that $h \in H(q)$ and therefore $\{h\} \cup H(h) \subseteq H(q)$. From Lemma 3.2 we have that $H(h) = H(x) - \{h\}$. Hence $H(x) \subseteq H(q)$. As we have assumed for contradiction that there is no crossing arc (v, u) such that $H(x) \not\subseteq H(u)$ it follows that the arc (p, q) can not be crossing. Hence the arc (p, q) must be backward. Since $(p, q) \in P_{r,t}$ the path $P_{r,t}[r, p]$ does not contain q and, therefore, (p, q) is relevant. Thus $(p, q) \in R(p)$ and $q \in H(p)$. We therefore have that $\{q\} \cup H(q) \subseteq H(p)$. Hence $\{q\} \cup H(x) \subseteq H(p)$. But it means that $|H(x)| < |H(p)|$ which contradicts that x was chosen such that $|H(x)| \geq |H(p)|$. ◀

3.1 Existence of Substitute-path for the Out-branching B_r^+

Now we define a substitute-path for the out-branching B_r^+ in the digraph $D = (V, A)$.

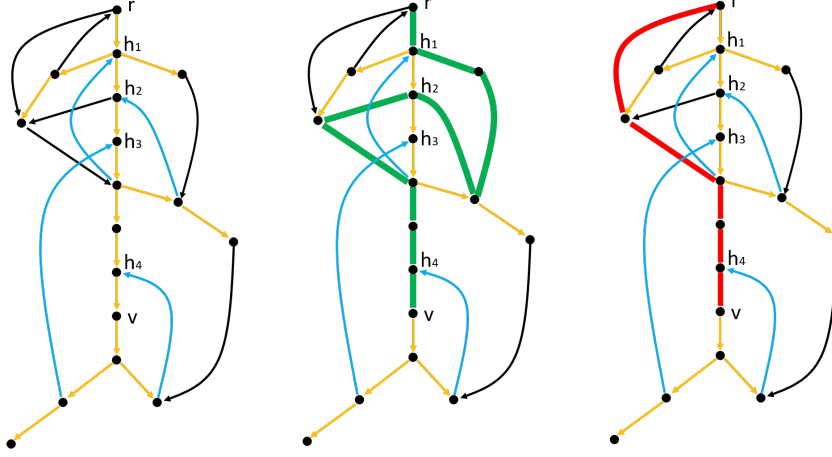
► **Definition 5.** Let u be a vertex in B_r^+ , then a **substitute-path** is a path $S_{r,u}$ from r to u in D such that

$$|J(u) \cap A(S_{r,u})| + |H(u) \setminus V(S_{r,u})| \geq \left\lceil \frac{|H(u)|}{2} \right\rceil. \quad (1)$$

Thus if there exists a substitute-path $S_{r,u}$ to a vertex u in B_r^+ then $S_{r,u}$ either contain at most half of the vertices in $H(u)$ or for each vertex from $H(u)$ it contains more than the half of the vertices in $H(u)$ it also contain an arc from $J(u)$. Let q be the number of vertices from $H(u)$ which are on a substitute-path $S_{r,u}$. It then follows that the number of arcs from $J(u)$ on $S_{r,u}$ is at least $q - \lceil \frac{|H(u)|}{2} \rceil$. Figure 4 shows two substitute-paths in an out-branching.

The following result may be of independent interests. Theorem 3.1 means that every vertex $v \in V$ is reachable from r by a substitute-path.

► **Theorem 3.1.** For the out-branching B_r^+ there exists a substitute-path to every $v \in V$.



■ **Figure 4** To the left, the digraph $D = (V, A)$ is shown. The arcs of the out-branching B_r^+ are shown in yellow, the arcs in $J(v)$ are blue, and the remaining arcs are black. In the middle and to the right, two paths from r to v are shown. One of the paths is green, the other is shown in red. Both paths are substitute-paths to v .

Proof. In this proof we will be modifying paths and, therefore, we will first consider two claims which deals with these modifications.

▷ **Claim 6.** For an arbitrary path $P_{r,v}$ in D the following inequality holds for every vertex $x \in V(P_{r,v})$

$$\begin{aligned} |J(v) \cap A(P_{r,v}[r, x])| + |H(v) \setminus V(P_{r,v}[r, x])| \\ \geq |J(v) \cap A(P_{r,v})| + |H(v) \setminus V(P_{r,v})|. \end{aligned} \quad (2)$$

Proof. As $P_{r,v}[r, x] \subseteq P_{r,v}$ we clearly have that $|J(v) \cap A(P_{r,v}[r, x])| \leq |J(v) \cap A(P_{r,v})|$. Let $j = |J(v) \cap A(P_{r,v})| - |J(v) \cap A(P_{r,v}[r, x])|$. For every arc $(t, h) \in J(v)$ which is also on the path $P_{r,v}$ it follows that $h \in H(v)$ and $h \in V(P_{r,v})$. Thus, the set $H(v) \setminus V(P_{r,v}[r, x])$ contains at least j vertices more than $H(v) \setminus V(P_{r,v})$. That is, $|H(v) \setminus V(P_{r,v}[r, x])| - j \geq |H(v) \setminus V(P_{r,v})|$. From this (2) follows. ◁

▷ **Claim 7.** Given a vertex $v \in B_r^+$, and two paths $P_{r,x}$ and $P_{x,u}$ in D which are disjoint on the vertex set $H(v) - \{x\}$ the following inequality holds for the walk $W = P_{r,x}P_{x,u}$ and for every (r, u) -path $P = P_{r,x}P_{x,u}$ obtained from W by deleting cycles.

$$\begin{aligned} |J(v) \cap A(W)| + |H(v) \setminus V(W)| \\ \leq |J(v) \cap A(P)| + |H(v) \setminus V(P)|. \end{aligned} \quad (3)$$

Proof. If $|J(v) \cap A(W)| > |J(v) \cap A(P)|$ then there must exist one or more cycles in W which contain arcs from $J(v)$. For each such cycle there must exist at least one vertex $z \in (V(P_{r,x}) \cap V(P_{x,u})) - \{x\}$. Since $P_{r,x}$ and $P_{x,u}$ are disjoint on the set $H(v) - \{x\}$ we have that $z \notin H(v)$. It means that for every arc $(t, h) \in J(v)$ contained in W but not in P the head h is not in P either. Therefore, for $j = |J(v) \cap A(W)| - |J(v) \cap A(P)|$ we have $|H(v) \setminus V(P)| - j \geq |H(v) \setminus V(W)|$ and Claim 7 follows. ◁

We are now ready to prove that there exists a substitute-path to every $v \in V$. We will prove this by induction over the cardinality of $H(v) = \mathcal{H}(J(v))$. Clearly, $|H(r)| = 0$ since no backwards arc goes over r . Hence there exists at least one vertex $v \in V$ for which $|H(v)| = 0$.

11:8 k -Distinct Branchings Admits a Polynomial Kernel

For every such vertex the path $B_r^+[r, v]$ is clearly a substitute-path by (1). Thus there exists a substitute-path to every $v \in V$ for which $|H(v)| = 0$. Now assume for some positive integer $i \leq n$ that for every vertex $v \in V$ where $|H(v)| < i$ it holds that there exists a substitute-path to v .

Let $u \in V$ be a vertex for which $|H(u)| = i$. If no such vertex exists in B_r^+ then clearly there exists a substitute-path to every vertex $u \in V$ where $|H(u)| = i$. Hence assume that u exists. Recall that $H(u) \subseteq V(B_r^+[r, u])$. Denote the vertices in $H(u)$ by h_0, h_1, \dots, h_{i-1} such that $B_r^+[r, u] = B_r^+[r, h_0]B_r^+[h_0, h_1] \cdots B_r^+[h_{i-2}, h_{i-1}]B_r^+[h_{i-1}, u]$.

▷ **Claim 8.** $H(h_j) = \{h_0, h_1, \dots, h_{j-1}\}$ for every $j \leq i - 1$.

Proof. From Lemma 3.2 it follows that $H(h_{i-1}) = \{h_0, h_1, \dots, h_{i-2}\}$. Now the claim follows from Lemma 3.2 by induction over $j \leq i - 1$. \triangleleft

Since $h_j \in B_r^+[r, h_{i-1}]$ for $j \leq i - 1$ no backwards arc with head in h_{i-1} can go over h_j . As $h_{i-1} \in H(u)$ this means that there exists a relevant arc which goes over u . Fix such an arc (t, h_{i-1}) .

▷ **Claim 9.** If there exists a path $P_{r,t}$ in D such that

$$|J(u) \cap A(P_{r,t})| + |H(u) \setminus V(P_{r,t})| \geq \lceil \frac{i}{2} \rceil. \quad (4)$$

then there is a substitute-path to u

Proof. Assume that there exists a path $P_{r,t}$ such that $|J(u) \cap A(P_{r,t})| + |H(u) \setminus V(P_{r,t})| \geq \lceil \frac{i}{2} \rceil$. Let x be the first vertex on $P_{r,t}$ which is also on the path $B_r^+[h_{i-1}, t]$. If $x = u$ then the path $P_{r,t}[r, u]$ will be a substitute-path to u as $|J(u) \cap A(P_{r,t})| + |H(u) \setminus V(P_{r,t})| \leq |J(u) \cap A(P_{r,t}[r, u])| + |H(u) \setminus V(P_{r,t}[r, u])|$ by Claim 6. Therefore, we may assume that $x \neq u$. We will consider the two cases: $x \in B_r^+[h_{i-1}, u]$ or $x \in B_r^+[u, t]$. By claim 6 it holds that

$$\begin{aligned} & |J(u) \cap A(P_{r,t}[r, x])| + |H(u) \setminus V(P_{r,t}[r, x])| \\ & \geq |J(u) \cap A(P_{r,t})| + |H(u) \setminus V(P_{r,t})| \geq \lceil \frac{i}{2} \rceil. \end{aligned} \quad (5)$$

If $x \in B_r^+[h_{i-1}, u]$ then for the walk $W'_{r,u} = P_{r,t}[r, x]B_r^+[x, u]$ it clearly holds that $|J(u) \cap A(P_{r,t}[r, x])| + |H(u) \setminus V(P_{r,t}[r, x])| = |J(u) \cap A(W'_{r,u})| + |H(u) \setminus V(W'_{r,u})|$. For the path $P'_{r,u} = P_{r,t}[r, x]B_r^+[x, u]$ we therefore have by Claim 7 that $|J(u) \cap A(P'_{r,u})| + |H(u) \setminus V(P'_{r,u})| \geq |J(u) \cap A(P_{r,t}[r, x])| + |H(u) \setminus V(P_{r,t}[r, x])|$. From (5) it therefore follows that $|J(u) \cap A(P'_{r,u})| + |H(u) \setminus V(P'_{r,u})| \geq \lceil \frac{i}{2} \rceil$. Hence $P'_{r,u}$ is a substitute-path to u . Now consider the second case where $x \in B_r^+[u, t]$. Consider the (x, u) -walk $P''_{x,u} = B_r^+[x, t](t, h_{i-1})B_r^+[h_{i-1}, u]$ and observe that $J(u) \cap A(P''_{x,u}) = (t, h_{i-1})$ and $H(u) \cap V(P''_{x,u}) = h_{i-1}$. It means that $P''_{x,u}$ is disjoint from $H(u) - \{h_{i-1}\}$. As x is the first vertex on $P_{r,t}$ which is on $B_r^+[h_{i-1}, t]$ and $x \in B_r^+[u, t]$ it holds that $h_{i-1} \notin P_{r,t}[r, x]$ and $P_{r,t}[r, x]$ and $P''_{x,u}$ are disjoint except in x . Now consider the $P^*_{r,u} = P_{r,t}[r, x]P''_{x,u}$ and observe that

$$\begin{aligned} & |J(u) \cap A(P^*_{r,u})| + |H(u) \setminus V(P^*_{r,u})| \\ & = |J(v) \cap A(P_{r,t}[r, x])| + 1 + |H(v) \setminus V(P_{r,t}[r, x])| - 1 \\ & = |J(v) \cap A(P_{r,t}[r, x])| + |H(v) \setminus V(P_{r,t}[r, x])| \end{aligned}$$

From (5) it therefore follows that $|J(u) \cap A(P^*_{r,u})| + |H(u) \setminus V(P^*_{r,u})| \geq \lceil \frac{i}{2} \rceil$. Thus $P^*_{r,u}$ is a substitute-path to u . \triangleleft

Now we prove that there always exists a path such that (4) holds. Recall by the definition of a relevant arc, that the relevant arc (t, h_{i-1}) goes over t and there exists a path $P_{r,t}$ in D such that $h_{i-1} \notin P_{r,t}$. If $|H(u) \setminus V(P_{r,t})| = i$ then (4) clearly holds. Hence we may assume that there exists a vertex $h_j \in H(u) \cap V(P_{r,t})$. Now let h_l be the last such vertex on the path $P_{r,t}$, that is, no vertex $h_j \in H(u) - \{h_l\}$ is on the path $P_{r,t}[h_l, t]$. Observe that since $P_{r,t}$ does not contain h_{i-1} we have

$$l \leq i - 2 \quad (6)$$

By Claim 8, $|H(h_j)| = j$ for every $j \leq i - 1$. By the induction assumption it therefore follows that there is a substitute-path S_{r,h_j} to every $h_j \in H(u)$.

▷ **Claim 10.** For every substitute-path to h_l , S_{r,h_l} , either there exists a vertex $h_j \in S_{r,h_l}$ such that $j > l$ or (4) holds.

Proof. Assume that there exists a substitute-path S_{r,h_l} to h_l such that there is no vertex $h_j \in S_{r,h_l}$ for $j > l$. Observe that since S_{r,h_l} is a substitute-path it holds that $|J(h_l) \cap A(S_{r,h_l})| + |H(h_l) \setminus V(S_{r,h_l})| \geq \lceil \frac{l}{2} \rceil$. Now consider the walk $W'_{r,t} = S_{r,h_l} P_{r,t}[h_l, t]$. The path S_{r,h_l} does not contain any heads $h_j \in H(u)$ for $j > l$ and $P_{r,t}[h_l, t]$ does not contain any heads $h_j \in H(u)$ for $j \neq l$. Hence we can deduce that:

$$\begin{aligned} & |J(u) \cap A(W'_{r,t})| + |H(u) \setminus V(W'_{r,t})| \\ & \geq |J(h_l) \cap A(S_{r,h_l})| + |H(h_l) \setminus V(S_{r,h_l})| + |\{h_j | l < j < i\}| \\ & = |J(h_l) \cap A(S_{r,h_l})| + |H(h_l) \setminus V(S_{r,h_l})| + i - l - 1 \\ & \geq \lceil \frac{l}{2} \rceil + i - l - 1 \end{aligned}$$

From this and Claim 7 we have the following inequality for the path $P'_{r,t} = S_{r,h_l} P_{r,t}[h_l, t]$.

$$|J(u) \cap A(P'_{r,t})| + |H(u) \setminus V(P'_{r,t})| \geq \lceil \frac{l}{2} \rceil + i - l - 1 \quad (7)$$

From (6) it follows that $l \leq i - 2$ and since $l \geq 0$ it also holds that $i \geq 2$. These observations together with (7) give us the following inequality.

$$|J(u) \cap A(P'_{r,t})| + |H(u) \setminus V(P'_{r,t})| \geq \lceil \frac{l}{2} \rceil + i - l - 1 \geq \lceil \frac{i}{2} \rceil$$

It means that $P'_{r,t}$ is a path making (4) true. Hence if (4) does not hold we may assume that for every substitute-path S_{r,h_l} to h_l there exists a $j > l$ such that $h_j \in S_{r,h_l}$. ◁

Let S_{r,h_j} be a substitute-path to $h_j \in H(u)$. If there exists a vertex $h_p \in S_{r,h_j}$ such that $p > j$ we will call the first vertex h_p with $p > j$ on the path S_{r,h_j} the **first exceeding head**. Recall that we are proving that a path exists such that (4) holds. Assume for contradiction that no such path exists. Thus by Claim 10 it follows that for every substitute-path S_{r,h_l} to h_l there is a vertex $h_j \in S_{r,h_l}$ for $j > l$, that is, every substitute-path S_{r,h_l} will contain a first exceeding head h_j such that $j > l$. Now let $k \leq l$ be the smallest integer such that for every substitute-path S_{r,h_k} the first exceeding head h_j has $j > l$. Observe that l fulfills this property and therefore k exists. Let S'_{r,h_k} be a substitute-path to h_k and h_j the first exceeding head on this path. Observe that $S'_{r,h_k}[r, h_j]$ is disjoint from the heads $\{h_k, h_{k+1}, \dots, h_{j-1}\}$. Therefore we have:

$$\begin{aligned} & |J(h_j) \cap A(S'_{r,h_k}[r, h_j])| + |H(h_j) \setminus V(S'_{r,h_k}[r, h_j])| \\ & \geq |J(h_k) \cap A(S'_{r,h_k})| + |H(h_k) \setminus V(S'_{r,h_k})| + |\{h_p | k \leq p < j\}| \\ & \geq |J(h_k) \cap A(S'_{r,h_k})| + |H(h_k) \setminus V(S'_{r,h_k})| + j - k \end{aligned} \quad (8)$$

11:10 k -Distinct Branchings Admits a Polynomial Kernel

As S'_{r,h_k} is a substitute-path to h_k and $|H(h_k)| = k$ we have that $|J(h_k) \cap A(S'_{r,h_k})| + |H(h_k) \setminus V(S'_{r,h_k})| \geq \lceil \frac{k}{2} \rceil$. When we combine this with (8) we obtain the following inequality:

$$|J(h_j) \cap A(S'_{r,h_k}[r, h_j])| + |H(h_j) \setminus V(S'_{r,h_k}[r, h_j])| \geq \lceil \frac{k}{2} \rceil + j - k \quad (9)$$

Now consider the (r, t) -path $P^* = S'_{r,h_k}[r, h_j]B_r^+[h_j, t]$. Recall that the path $B_r^+[h_j, t]$ is disjoint from every head h_p for $p < j$ and $S'_{r,h_k}[r, h_j]$. It therefore follow from (9) that.

$$\begin{aligned} & |J(u) \cap A(P^*)| + |H(u) \setminus V(P^*)| \\ & \geq |J(h_j) \cap A(S'_{r,h_k}[r, h_j])| + |H(h_j) \setminus V(S'_{r,h_k}[r, h_j])| \\ & \geq \lceil \frac{k}{2} \rceil + j - k \end{aligned} \quad (10)$$

From (10) and the fact that $l + 1 \leq j$ we obtain the following inequality.

$$|J(u) \cap A(P^*)| + |H(u) \setminus V(P^*)| \geq l + 1 - \lfloor \frac{k}{2} \rfloor \quad (11)$$

From the assumption that (4) does not hold it follows that $|J(u) \cap A(P^*[r, t])| + |H(u) \setminus V(P^*[r, t])| < \lceil \frac{i}{2} \rceil$. Therefore, we have the following observation from (11).

► **Observation 11.** $l + 1 - \lfloor \frac{k}{2} \rfloor < \lceil \frac{i}{2} \rceil$.

Recall that the path $B_r^+[r, h_0]$ is a substitute-path to h_0 and as every substitute-path to h_k has a first exceeding head h_p such that $p > l$ it must hold that $k > 0$. Furthermore, recall that k was chosen as the smallest integer such that the first exceeding head h_p on every substitute-path to h_k had $p > l$. Hence there exist a substitute-path $S^*_{r,h_{k-1}}$ such that either there is no exceeding head or for the first exceeding head h_p it holds that $k - 1 < p \leq l$. If there is no exceeding head let $p = k - 1$ and otherwise let h_p be the first exceeding head. Consider $S^*_{r,h_{k-1}}[r, h_p]$. Observe by Claim 6 and the fact that $S^*_{r,h_{k-1}}$ is a substitute-path that

$$|J(h_{k-1}) \cap A(S^*_{r,h_{k-1}}[r, h_p])| + |H(h_{k-1}) \setminus V(S^*_{r,h_{k-1}}[r, h_p])| \geq \lceil \frac{k-1}{2} \rceil$$

Now consider the (r, t) -walk $W'_{r,t} = S^*_{r,h_{k-1}}[r, h_p]B_r^+[h_p, h_l]P_{r,t}[h_l, t]$. Note that $S^*_{r,h_{k-1}}[r, h_p]$ is disjoint from every head h_q for $q > p$, $B_r^+[h_p, h_l]$ is disjoint from every head h_q for $q < p$ and $q > l$, and furthermore $P_{r,t}[h_l, t]$ is disjoint from $H(u) - \{h_l\}$. Hence:

$$\begin{aligned} & |J(u) \cap A(W')| + |H(u) \setminus V(W')| \\ & = |J(h_{k-1}) \cap A(S^*_{r,h_{k-1}}[r, h_p])| + |H(h_{k-1}) \setminus V([S^*_{r,h_{k-1}}[r, h_p])| + |\{h_q | l < q < i\}| \\ & = |J(h_{k-1}) \cap A(S^*_{r,h_{k-1}}[r, h_p])| + |H(h_{k-1}) \setminus V(S^*_{r,h_{k-1}}[r, h_p])| + i - 1 - l \end{aligned} \quad (12)$$

Consider the (r, t) -path $P' = S^*_{r,h_{k-1}}[r, h_p]B_r^+[h_p, h_l]P_{r,t}[h_l, t]$. Then from (12), claim 7 and the fact that $S^*_{r,h_{k-1}}$ is a substitute-path to h_{k-1} we obtain the following:

$$\begin{aligned} & |J(u) \cap A(P')| + |H(u) \setminus V(P')| \\ & \geq |J(u) \cap A(W')| + |H(u) \setminus V(W')| \\ & \geq \lceil \frac{k-1}{2} \rceil + i - 1 - l \end{aligned} \quad (13)$$

By the assumption that (4) does not hold we have $|J(u) \cap A(P')| + |H(u) \setminus V(P')| < \lceil \frac{i}{2} \rceil$. This combined with (13) gives us:

$$\lceil \frac{i}{2} \rceil > \lceil \frac{k-1}{2} \rceil + i - 1 - l \quad \Leftrightarrow \quad l + 1 - \lceil \frac{k-1}{2} \rceil > \lceil \frac{i}{2} \rceil \quad (14)$$

Combining Observation 11 with (14) and the fact that $\lceil \frac{k-1}{2} \rceil = \lfloor \frac{k}{2} \rfloor$ we obtain the following inequality:

$$\lfloor \frac{i}{2} \rfloor < l + 1 - \lfloor \frac{k}{2} \rfloor < \lceil \frac{i}{2} \rceil \quad (15)$$

Now observe that $l + 1 - \lfloor \frac{k}{2} \rfloor$ is an integer and from (15) we therefore have a contradiction. Hence either the path P^* or the path P' makes (4) true. It means that there exists a path making (4) true. Thus by Claim 9 there exists a substitute-path to u .

Recall that u was an arbitrary fixed vertex $x \in V$ for which $|H(x)| = i$. Thus we can conclude that for every vertex $x \in V$ where $|H(x)| = i$ there exists a substitute path to x . This concludes the induction step over for every vertex $x \in V$ where $|H(x)| = i$. Thus we can conclude that for every vertex $x \in V$ where $|H(x)| = i$ for $i \in [0, |V|]$ there exists a substitute-path. As $|H(x)| \leq n$ for every $x \in V$, the proof is complete. \blacktriangleleft

3.2 Finding Desired Paths in Polynomial Time

We now obtain the following lemma about finding a substitute-path to a vertex $v \in V$.

► **Lemma 3.4** (\star). *There exists a polynomial time algorithm which given an out-branching B_r^+ in $D = (V, A)$ and a vertex $v \in V$ finds a substitute-path to v .*

► **Lemma 3.5** (\star). *For every vertex $u \in V(B_r^+)$ there exists a path $P_{r,u}$ from r to u such that*

$$|R(u) \cap A(P_{r,u})| + |\mathcal{H}(R(u)) \setminus V(P_{r,u})| \geq \left\lceil \frac{|\mathcal{H}(R(u))|}{2} \right\rceil \quad (16)$$

and it can be found in polynomial time

► **Lemma 3.6** (\star). *For every vertex $v \in V(B_r^+)$ let $R_v \in \{J(v), R(v)\}$. If there exists a path $P_{r,v}$ such that*

$$|R_v \cap A(P_{r,v})| + |\mathcal{H}(R_v) \setminus V(P_{r,v})| \geq \left\lceil \frac{|\mathcal{H}(R_v)|}{2} \right\rceil \quad (17)$$

then there exists an out-branching \widehat{B}_r^+ such that $|R_v \cap A(\widehat{B}_r^+)| \geq \left\lceil \frac{|\mathcal{H}(R_v)|}{2} \right\rceil$, and \widehat{B}_r^+ can be found in polynomial time.

4 Backward arcs, Crossing Arcs and Substitute-paths with respect to In-branchings

For in-branchings we have similar definitions and results as we have for out-branchings. Given a digraph $D = (V, A)$ which contains a fixed in-branching B_r^- we can create a corresponding digraph $D' = (V, A')$ and an out-branching B_r^+ by creating an arc (v, u) in A' for each arc $(u, v) \in A$. That is, the direction of the arcs in A are flipped. Observe that the in-branching B_r^- in D will correspond to a fixed out-branching B_r^+ in D' . Due to this one-to-one correspondence all the results given in Section 3 regarding a fixed out-branching will be turned into equivalent statements about a fixed in-branching.

5

A kernel for ROOTED k -DISTINCT BRANCHINGS

In this section we prove the existence of an $\mathcal{O}(k^2)$ -vertex kernel for the problem ROOTED k -DISTINCT BRANCHINGS. The proof of the next lemma will give the desired Theorem 1.1.

► **Lemma 5.1.** *Given an instance $(D = (V, A), k, s, t)$ of R - k -DB; In polynomial time we can either find a kernel (D', k, s', t') such that $|V(D')| < 16k^2 + 5k = \mathcal{O}(k^2)$ or determine the answer to the instance.*

Proof. In linear time it can be determined if an out-branching B_s^+ and an in-branching B_t^- exist. If one of these do not exist then clearly (D, k, s, t) is a no-instance. Hence assume that both exist. At any time during the proof let \mathcal{C}^+ , \mathcal{R}^+ , and \mathcal{I}^+ , respectively, denote the crossing, relevant, and irrelevant arcs wrt.. B_s^+ . Similarly, let \mathcal{C}^- , \mathcal{R}^- , and \mathcal{I}^- , respectively, denote the crossing, relevant, and irrelevant arcs wrt.. B_t^- . Moreover, at all times during the proof let $F = A(D) \setminus (A(B_s^+) \cup A(B_t^-))$ be defined as the **free arcs** and at all times let $E^+ = A(B_s^+) \setminus A(B_t^-)$, $E^- = A(B_t^-) \setminus A(B_s^+)$ denote the **exclusive set**, respectively, for B_s^+ and B_t^- . Note that $|E^+| = |E^-|$. Now we execute the following procedure.

- **Procedure 5.1.** *Change B_s^+ and B_t^- as follows until no longer possible or $|E^+| \geq k$.*
1. *If there exists an arc $(u, v) \in F \cap \mathcal{C}^+$ such that $(\mathcal{P}_{B_s^+}(v), v) \notin E^+$, then remove $(\mathcal{P}_{B_s^+}(v), v)$ from B_s^+ and insert the arc (u, v) into B_s^+ .*
 2. *If there exists an arc $(u, v) \in F \cap \mathcal{C}^-$ such that $(u, \mathcal{P}_{B_t^-}(u)) \notin E^-$, then remove $(u, \mathcal{P}_{B_t^-}(u))$ from B_t^- and insert the arc (u, v) into B_t^- .*

► **Lemma 5.2** (\star) . *Procedure 5.1 can be executed in polynomial time.*

If we have not found a solution when Procedure 5.1 terminates, then we have

$$|E^+| = |E^-| < k. \quad (18)$$

Since $|E^+| < k$ and we can not change B_s^+ further in Procedure 5.1 we have that for every crossing arc $(u, v) \in \mathcal{C}^+$ either $(u, v) \in A(B_t^-)$ or $(\mathcal{P}_{B_s^+}(v), v) \in E^+$. As \mathcal{C}^+ is disjoint from B_s^+ it follows that every arc $(u, v) \in A(B_t^-) \cap \mathcal{C}^+$ is contained in E^- . Hence there are less than k such arcs. For the arcs $(u, v) \in \mathcal{C}^+$ where $(\mathcal{P}_{B_s^+}(v), v) \in E^+$ there is less than k different heads as $|E^+| < k$. Therefore, there must be less than $2k$ different heads for the crossing arcs of B_s^+ , that is, $|\mathcal{H}(\mathcal{C}^+)| < 2k$. Similarly, we have that there are less than $2k$ different tails of \mathcal{C}^- , that is, $|\mathcal{T}(\mathcal{C}^-)| < 2k$.

► **Observation 12.** $|\mathcal{H}(\mathcal{C}^+)| < 2k$ and $|\mathcal{T}(\mathcal{C}^-)| < 2k$

A vertex $v \in V$ is a **Type 1** vertex if it is the tail of an arc $(v, u) \in \mathcal{R}^- \cup \mathcal{C}^-$ and a **Type 2** vertex if it is the head of an arc $(u, v) \in \mathcal{R}^+ \cup \mathcal{C}^+$. Note that a vertex $v \in V$ can be both a Type 1 and a Type 2 vertex. Now we have the following reduction rule.

► **Reduction Rule 5.1.** *As long as there exists an arc $(u, v) \in A(B_s^+) \cap A(B_t^-)$ such that u is not a Type 1 vertex and v is not a Type 2 vertex contract (u, v) into one vertex.*

► **Lemma 5.3** (\star) . *Reduction Rule 5.1 is safe and can be applied in polynomial time.*

Let D_R , B_s^+ and B_t^- be the digraph and branchings that we have obtained after performing Reduction 5.1. We now have the following lemma.

► **Lemma 5.4.** *If $|V(D_R)| \geq 16k^2 + 5k$ then a solution exists.*

Proof. Assume $|V(D_R)| \geq 16k^2 + 5k$. As B_s^+ is an out-branching we have $|A(B_s^+)| = |V(D_R)| - 1$. For every arc $(u, v) \in A(B_s^+)$ it holds that either $(u, v) \notin A(B_t^-)$, u is a Type 1 vertex or v is a Type 2 vertex. By (18) we see that $|A(B_s^+) \setminus A(B_t^-)| < k$. It means that $|A(B_s^+) \cap A(B_t^-)| \geq |V(D_R)| - k \geq 2 \cdot (8k^2 + 2k)$. For every arc $(u, v) \in A(B_s^+) \cap A(B_t^-)$ we have that u is a Type 1 vertex or v is a Type 2 vertex. Hence we can conclude that either there is $8k^2 + 2k$ vertices of Type 1 or $8k^2 + 2k$ vertices of Type 2 (or both). In the following we will only explicitly give the proof that a solution exists if there are at least $8k^2 + 2k$ vertices of Type 2 as the proof for a solution exists if there are at least $8k^2 + 2k$ vertices of Type 1 will follow from the symmetry between $B_s^+, C^+, \mathcal{R}^+, \mathcal{I}^+$ and $B_t^-, C^-, \mathcal{R}^-, \mathcal{I}^-$. Assume therefore that there are at least $8k^2 + 2k$ vertices of Type 2. Recall that it means, there are at least $8k^2 + 2k$ different vertices which are heads of the arcs $\mathcal{R}^+ \cup C^+$. By Observation 12 there are less than $2k$ vertices which are heads of C^+ . Consequently, the number of vertices which are heads of \mathcal{R}^+ must be larger than $8k^2$. Clearly, $|\mathcal{H}(\mathcal{R}^+)| = |\cup_{v \in V} \mathcal{H}(J(v))|$ and we therefore have $8k^2 \leq |\mathcal{H}(\mathcal{R}^+)| = |\cup_{v \in V} \mathcal{H}(J(v))|$. By Lemma 3.3 we have that for every $v \in V$ it holds that $\mathcal{H}(J(v)) \subseteq \mathcal{H}(J(u))$ for some arc $(v, u) \in C^+$. We therefore have that: $|\cup_{(v,u) \in C^+} \mathcal{H}(J(u))| = |\cup_{v \in V} \mathcal{H}(J(v))| \geq 8k^2$. By Observation 12 there are at most $2k$ heads of the arcs in C^+ so there must exist at least one arc $(u, v) \in C^+$ such that $|\mathcal{H}(J(v))| \geq 4k$. Fix (u, v) to be such an arc. By Theorem 3.1 there exist a substitute-path $S[s, v]$ in D such that $|J(v) \cap A(S[s, v])| + |\mathcal{H}(J(v)) \setminus A(S[s, v])| \geq \left\lceil \frac{|\mathcal{H}(J(v))|}{2} \right\rceil$ and therefore by from Lemma 3.6 there exists an out-branching \hat{B}_s^+ such that $|J(v) \cap A(\hat{B}_s^+)| \geq \left\lceil \frac{|\mathcal{H}(J(v))|}{2} \right\rceil \geq 2k$. Observe that B_s^+ is disjoint from $J(v)$ and therefore if $|J(v) \cap A(B_t^-)| \geq k$ then B_s^+ and B_t^- would have been a solution. If $|J(v) \cap A(B_t^-)| < k$ then $|A(\hat{B}_s^+) \setminus A(B_t^-)| \geq k$ and \hat{B}_s^+ and B_t^- is a solution. Hence we conclude that if the number of different heads of R^+ is larger than $4k^2$ a solution exists. \blacktriangleleft

From Lemma 5.4 we can conclude that after applying Reduction Rule 5.1 it either holds that we have a solution or $|V(D_R)| < 16k^2 + 5k$. In the first case we have a solution in the latter we have a kernel of size $\mathcal{O}(k^2)$. It therefore only remains to argue that the solution or the kernel can be found in polynomial time. For an instance $(D = (V, A), k, s, t)$ of R- k -DB it is possible in polynomial time to decide if an out-branching B_s^+ and B_t^- in D exists and find them. By Claim 5.2, and Claim 5.3 we can execute the Procedure 5.1 and afterwards apply Reduction Rule 5.1 in polynomial time. Furthermore, it is polynomial to decide if the resulting digraph has at least $16k^2 + 5k$ vertices and applying Lemma 5.4. Hence in polynomial time we can either find a kernel with a vertex set of size less than $16k^2 + 5k$ or determine the answer to the instance. \blacktriangleleft

6 Conclusion

In this paper, we studied the problem of deciding if a digraph $D = (V, A)$, contains an in- and out-branching rooted at specific vertices s and t , such that the in- and out-branching are distinct on at least k arcs. Before this paper, it was not known if the problem admitted a polynomial kernel, and the best known complexity for solving the problem was $2^{\mathcal{O}(k^2 \log^2 k)} n^{\mathcal{O}(1)}$. We designed a polynomial kernel for the problem with $\mathcal{O}(k^2)$ vertices and found an algorithm with the complexity $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$. To obtain these results, we defined the concept of substitute-paths in out- and in-branchings. This graph-theoretical concept might be useful for obtaining other results on problems regarding in- and out-branchings. It is still open whether there exists a kernel with $\mathcal{O}(k)$ vertices. We believe that using representative set approach

applied for obtaining exact exponential time algorithm for finding a strongly connected subgraph of a given digraph with minimum number of arcs [20], it seem possible to get an algorithm for R-*k*-DB running in time $2^{\mathcal{O}(k)} + n^{\mathcal{O}(1)}$. Making this work seems an interesting direction to pursue.

References

- 1 Noga Alon, Fedor V. Fomin, Gregory Gutin, Michael Krivelevich, and Saket Saurabh. Spanning directed trees with many leaves. *SIAM J. Discrete Math.*, 23(1):466–476, 2009. doi:10.1137/070710494.
- 2 J. Bang-Jensen. Edge-disjoint in- and out-branchings in tournaments and related path problems. *J. Combin. Theory Ser. B*, 51(1):1–23, 1991.
- 3 J. Bang-Jensen and J. Huang. Decomposing locally semicomplete digraphs into strong spanning subdigraphs. *J. Combin. Theory Ser. B*, 102:701–714, 2010.
- 4 J. Bang-Jensen, S. Saurabh, and S. Simonsen. Parameterized algorithms for non-separating trees and branchings in digraphs. *Algorithmica*, 76(1):279–296, 2016.
- 5 J. Bang-Jensen and S. Simonsen. Arc-disjoint paths and trees in 2-regular digraphs. *Discrete Applied Mathematics*, 161(16-17):2724–2730, 2013.
- 6 J. Bang-Jensen, S. Thomassé, and A. Yeo. Small degree out-branchings. *J. Graph Theory*, 42(4):297–307, 2003.
- 7 J. Bang-Jensen and A. Yeo. The minimum spanning strong subdigraph problem is fixed parameter tractable. *Discrete Appl. Math.*, 156:2924–2929, 2008.
- 8 J. Bang-Jensen and A. Yeo. Arc-disjoint spanning sub(di)graphs in digraphs. *Theor. Comput. Sci.*, 438:48–54, 2012.
- 9 K. Bérczi, S. Fujishige, and N. Kamiyama. A linear-time algorithm to find a pair of arc-disjoint spanning in-arborescence and out-arborescence in a directed acyclic graph. *Inform. Process. Lett.*, 109(23-24):1227–1231, 2009.
- 10 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshantov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Transactions on Algorithms*, 8(4):38, 2012. doi:10.1145/2344422.2344428.
- 11 Nathann Cohen, Fedor V. Fomin, Gregory 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 M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 13 J. Daligault, G. Gutin, E.J. Kim, and A. Yeo. FPT algorithms and kernels for the directed *k*-leaf problem. *J. Comput. Syst. Sci.*, 76:144–152, 2010.
- 14 Frederic Dorn, Fedor V. Fomin, Daniel Lokshantov, Venkatesh Raman, and Saket Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In *STACS*, volume 5, pages 251–262, 2010. doi:10.4230/LIPIcs.STACS.2010.2459.
- 15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 16 J. Edmonds. Edge-disjoint branchings. In *Combinatorial Algorithms*, pages 91–96. Academic Press, 1973.
- 17 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.
- 18 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Stéphan Thomassé. A linear vertex kernel for maximum internal spanning tree. *J. Comput. Syst. Sci.*, 79(1):1–6, 2013. doi:10.1016/j.jcss.2012.03.004.
- 19 Fedor V. Fomin, Fabrizio Grandoni, Daniel Lokshantov, 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.

- 20 Fedor V. Fomin, Daniel Lokshantov, 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.
- 21 Fedor V Fomin, Daniel Lokshantov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 22 Gregory Z. Gutin, Felix Reidl, and Magnus Wahlström. k -distinct in- and out-branchings in digraphs. *J. Comput. Syst. Sci.*, 95:86–97, 2018.
- 23 J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. *Algorithmica*, 61(4):882–897, 2011.
- 24 R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, Oxford, 2006.

Incremental Edge Orientation in Forests

Michael A. Bender ✉

Stony Brook University, NY, USA

Tsvi Kopelowitz ✉

Bar-Ilan University, Ramat Gan, Israel

William Kuszmaul ✉

Massachusetts Institute of Technology, Cambridge, MA, USA

Ely Porat ✉

Bar-Ilan University, Ramat Gan, Israel

Clifford Stein ✉

Columbia University, New York, NY, USA

Abstract

For any forest $G = (V, E)$ it is possible to orient the edges E so that no vertex in V has out-degree greater than 1. This paper considers the incremental edge-orientation problem, in which the edges E arrive over time and the algorithm must maintain a low-out-degree edge orientation at all times. We give an algorithm that maintains a maximum out-degree of 3 while flipping at most $O(\log \log n)$ edge orientations per edge insertion, with high probability in n . The algorithm requires worst-case time $O(\log n \log \log n)$ per insertion, and takes amortized time $O(1)$. The previous state of the art required up to $O(\log n / \log \log n)$ edge flips per insertion.

We then apply our edge-orientation results to the problem of dynamic Cuckoo hashing. The problem of designing simple families \mathcal{H} of hash functions that are compatible with Cuckoo hashing has received extensive attention. These families \mathcal{H} are known to satisfy *static guarantees*, but do not come typically with *dynamic guarantees* for the running time of inserts and deletes. We show how to transform static guarantees (for 1-associativity) into near-state-of-the-art dynamic guarantees (for $O(1)$ -associativity) in a black-box fashion. Rather than relying on the family \mathcal{H} to supply randomness, as in past work, we instead rely on randomness within our table-maintenance algorithm.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Dynamic graph algorithms

Keywords and phrases edge orientation, graph algorithms, Cuckoo hashing, hash functions

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.12

Related Version *Full Version:* <http://arxiv.org/abs/2107.01250>

Funding Supported in part by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064). This research was sponsored in part by National Science Foundation Grants XPS-1533644 and CCF-1930579, CCF-1714818, and CCF-1822809, CCF-2106827, CCF-1725543, CSR-1763680, CCF-1716252, and CNS-1938709. The research was also sponsored in part by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. 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 United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.



© Michael A. Bender, Tsvi Kopelowitz, William Kuszmaul, Ely Porat, and Clifford Stein; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 12; pp. 12:1–12:18

Leibniz International Proceedings in Informatics



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

1 Introduction

The general problem of maintaining low-out-degree edge orientations of graphs has been widely studied and has found a broad range of applications throughout algorithms (see, e.g., work on sparse graph representations [10], maximal matchings [7–9, 18, 20, 26], dynamic matrix-by-vector multiplication [20], etc.). However, some of the most basic and fundamental versions of the graph-orientation problem have remained unanswered.

This paper considers the problem of incremental edge orientation in forests. Consider a sequence of edges e_1, e_2, \dots, e_{n-1} that arrive over time, collectively forming a tree. As the edges arrive, one must maintain an **orientation** of the edges (i.e., to assign a direction to each edge) so that no vertex ever has out-degree greater than $O(1)$. The orientation can be updated over time, meaning that orientations of old edges can be flipped in order to make room for the newly inserted edges. The goal is achieve out-degree $O(1)$ while flipping as few edges as possible per new edge arrival.

Forests represent the best possible case for edge orientation: it is always possible to construct an orientation with maximum out-degree 1. But, even in this seemingly simple case, no algorithms are known that achieve better than $O(\log / \log \log n)$ edge flips per edge insertion [20]. A central result of this paper is that, by using randomized and intentionally non-greedy edge-flipping one can do substantially better, achieving $O(\log \log n)$ edges flips per insertion.

A warmup: two simple algorithms

As a warmup let us consider two simple algorithms for incremental edge-orientation in forests.

The first algorithm never flips any edges but allows the maximum out-degree of each vertex to be as high as $O(\log n)$. When an edge (u, v) is added to the graph, the algorithm examines the connected components T_u and T_v that are being connected by the edge, and determines which component is larger (say, $|T_v| \geq |T_u|$). The algorithm then orients the edge from u to v , so that it is directed out of the smaller component. Since the new edge is always added to a vertex whose connected component at least doubles in size, the maximum out-degree is $\lceil \log n \rceil$.

The second algorithm guarantees that the out-degree will always be 1, but at the cost of flipping more edges. As before, when (u, v) is added the algorithm orients the edge from u to v . If this increments the out-degree of u to 2, then the algorithm follows the directed path P in T_u starting from u (and such that the edge (u, v) is not part of P) until a vertex r with out-degree 0 is reached. The algorithm then flips the edge orientations on P , which increases the out-degree of r to be 1 and reduces the out-degree of u to be 1. Since every edge that is flipped is always part of a connected component that has just at least doubled in size, the number of times each edge is flipped (in total across all insertions) is at most $\lceil \log n \rceil$ and so the amortized time cost per insertion is $O(\log n)$.¹

These two algorithms sit on opposite sides of a tradeoff curve. In one case, we have maximum out-degree $O(\log n)$ and at most $O(1)$ edges flipped per insertion, and in the other we have maximum out-degree $O(1)$ and at most $O(\log n)$ (amortized) flips per insertion. This raises a natural question: *what is the optimal tradeoff curve between the maximum out-degree and the number of edges flipped per insertion?*

¹ By allowing for a maximum out-degree of 2, the bound of $O(\log n)$ on the number of edges flipped can be improved from being amortized to worst-case. In particular, for any vertex v there is always a (directed) path of length $O(\log n)$ to another vertex with out-degree 1 or less (going through vertices with out-degree 2); by flipping the edges in such a path, we can insert a new edge at the cost of only $O(\log n)$ flips.

Our results

We present an algorithm for incremental edge orientation in forests that satisfies the following guarantees with high probability in n :

- the maximum out-degree never exceeds 3;
- the maximum number of edges flipped per insertion is $O(\log \log n)$;
- the maximum time taken by any insertion is $O(\log n \log \log n)$;
- and the amortized time taken (and thus also the amortized number of edges flipped) per insertion is $O(1)$.

An interesting feature of this result is that the aforementioned tradeoff curve is actually quite different than it first seems: by increasing the maximum out-degree to 3 (instead of 2 or 1), we can decrease the maximum number of edges flipped per insertion all the way to $O(\log \log n)$.

In fact, a similar phenomenon happens on the other side of the tradeoff curve. For any ε , we show that it is possible to achieve a maximum out-degree of $\log^\varepsilon n + 1$ while only flipping $O(\varepsilon^{-1})$ edges per insertion. Notably, this means that, for any positive constant c , one can achieve out-degree $(\log n)^{1/c}$ with $O(1)$ edges flipped per insertion.

A key idea in achieving the guarantees above is to selectively leave vertices with low out-degrees “sprinkled” around the graph, thereby achieving an edge orientation that is amenable to future edges being added. Algorithmically, the main problem that our algorithm solves is that of high-degree vertices clustering in a “hotspot”, which could then force a single edge-insertion to invoke a large number of edge flips.

Related work on edge orientations

The general problem of maintaining low-out-degree orientations of dynamic graphs has served as a fundamental tool for many problems. Brodal and Fagerberg [10] used low-degree edge orientations to represent dynamic sparse graphs – by assigning each vertex only $O(1)$ edges for which it is responsible, one can then deterministically answer adjacency queries in $O(1)$ time. Low-degree edge orientations have also been used to maintain maximal matchings in dynamic graphs [7, 18, 20, 26], and this technique remains the state of the art for graphs with low arboricity. Other applications include dynamic matrix-by-vector multiplication [20], dynamic shortest-path queries in planar graphs [21], and approximate dynamic maximum matchings [8, 9].

The minimum out-degree attainable by any orientation of a graph is determined by the graph’s pseudo-arboricity α . As a result, the algorithmic usefulness of low out-degree orientations is most significant for graphs that have low pseudo-arboricity. This makes forests and pseudoforests (which are forests with one extra edge per component) especially interesting, since they represent the case of $\alpha = 1$ and thus always allow for an orientation with out-degree 1.

Whereas this paper focuses on edge orientation in incremental forests (and thus also incremental pseudoforests), past work has considered a slightly more general problem [7, 10, 18, 20], allowing for edge deletions in addition to edge insertions, and also considering dynamic graphs with pseudo-arboricities $\alpha > 1$. Brodal and Fagerberg gave an algorithm that achieved out-degree $O(\alpha)$ with amortized running time that is guaranteed to be constant competitive with that of any algorithm; they also showed that in the case of $\alpha \in O(1)$, it is possible to achieve constant out-degree with amortized time $O(1)$ per insertion and $O(\log n)$ per deletion [10]. For worst-case guarantees, on the other hand, the only algorithm known to achieve sub-logarithmic bounds for both out-degree and edges flipped per insertion is that of

Kopelowitz et al. [20], which achieves $O(\log n / \log \log n)$ for both, assuming $\alpha \in O(\sqrt{\log n})$. In the case of incremental forests, our results allow for us to improve substantially on this, achieving a worst-case bound of $O(\log \log n)$ edges flipped per insertion (with high probability) while supporting maximum out-degree $O(1)$. An interesting feature of our algorithm is that it is substantially different than any of the past algorithms, suggesting that the fully dynamic graph setting (with $\alpha > 1$) may warrant revisiting.

Our interest in the incremental forest case stems in part from its importance for a specific application: Cuckoo hashing. As we shall now discuss, our results on incremental edge orientation immediately yield a somewhat surprising result on Cuckoo hashing with dynamic guarantees.

1.1 An Application to Cuckoo Hashing: From Static to Dynamic Guarantees via Non-Greedy Eviction

A *s*-**associative Cuckoo hash table** [13, 23, 27, 28] consists of n **bins**, each of which has s **slots**, where s is a constant typically between 1 and 8 [23, 27]. Records are inserted into the table using two hash functions h_1, h_2 , each of which maps records to bins. The invariant that makes Cuckoo hashing special is that, if a record x is in the table, then x must reside in either bin $h_1(x)$ or $h_2(x)$. This invariant ensures that query operations *deterministically* run in time $O(1)$.

When a new record x is inserted into the table, there may not initially be room in either bin $h_1(x)$ or $h_2(x)$. In this case, x **kicks out** some record y_1 in either $h_1(x)$ or $h_2(x)$. This, in turn, forces y_1 to be inserted into the other bin b_2 to which y_1 is hashed. If bin b_2 *also* does not have space, then y_1 kicks out some record y_2 from bin b_2 , and so on. This causes what is known as a **kickout chain**. Formally, a kickout chain takes a sequence of records y_1, y_2, \dots, y_j that reside in bins b_1, b_2, \dots, b_j , respectively, and relocates those records to instead reside in bins b_2, b_3, \dots, b_{j+1} , respectively, where for each record y_i the bins b_i and b_{i+1} are the two bins to which h_1 and h_2 map y_i . The purpose of a kickout chain is to free up a slot in bin b_1 so that the newly inserted record can reside there. Although Cuckoo hashing guarantees constant-time queries, insertion operations can sometimes incur high latency due to long kickout chains.

The problem of designing simple hash-function families for Cuckoo hashing has received extensive attention [1, 4, 5, 11, 14, 15, 25, 27, 30]. Several natural (and widely used) families of hash functions are known *not* to work [11, 14], and it remains open whether there exists $k = o(\log n)$ for which k -independence suffices [24]. This has led researchers to design and analyze *specific families* of simple hash functions that have low independence but that, nonetheless, work well with Cuckoo hashing [1, 4, 5, 15, 25, 27, 30]. Notably, Cuckoo hashing has served as one of the main motivations for the intensive study of tabulation hash functions [1, 12, 29–31].

Work on hash-function families for cuckoo hashing [1, 4, 5, 15, 25, 27, 30] has focused on offering a **static guarantee**: for any set X of $O(n)$ records, there exists (with reasonably high probability) a valid 1-associative hash-table configuration that stores the records X . This guarantee is static in the sense that it does not say anything about the speed with which insertion and deletion operations can be performed.

On the other hand, if the hash functions are fully random, then a strong **dynamic guarantee** is known. Panigrahy [28] showed that, using bins of size two, insertions can be implemented to incur at most $\log \log n + O(1)$ kickouts, and to run in time at most $O(\log n)$, with high probability in n . Moreover, the expected time taken by each insertion is $O(1)$.

The use of bin sizes greater than one is essential here, as it gives the data structure algorithmic flexibility in choosing which record to evict from a bin. Panigrahy [28] uses breadth-first search in order to find the shortest possible kickout chain to a bin with a free slot. The fact that the hash functions h_1 and h_2 are fully random ensures that, with high probability, the search terminates within $O(\log n)$ steps, thereby finding a kickout chain of length $\log \log n + O(1)$.

If a family of hash functions has sufficiently strong randomness properties (e.g., the family of [15]) then one can likely recreate the guarantees of [28] by directly replicating the analysis. For other families of hash functions [1, 4, 5, 15, 25, 27, 30], however, it is unclear what sort of dynamic guarantees are or are not possible.

This raises a natural question: *does there exist a similar dynamic guarantee to that of [28] when the underlying hash functions are not fully random – in particular, if we know only that a hash family \mathcal{H} offers a static guarantee, but we know nothing else about the structure or behavior of hash functions in \mathcal{H} , is it possible to transform the static guarantee into a dynamic guarantee?*

Our results on Cuckoo hashing: a static-to-dynamic transformation

We answer this question in the affirmative by presenting a new algorithm, the Dancing-Kickout Algorithm, for selecting kickout chains during insertions in a Cuckoo hash table. Given any hash family \mathcal{H} that offers a 1-associative static guarantee, we show that the same hash family can be used to offer an $O(1)$ -associative dynamic guarantee. In particular, the Dancing-Kickout Algorithm supports both insertions and deletions with the following promise: as long as the static guarantee for \mathcal{H} has not failed, then with high probability, each insertion/deletion incurs at most $O(\log \log n)$ kickouts, has amortized time (and therefore number of kickouts) $O(1)$, and takes time at most $O(\log n \log \log n)$. We also extend our results to consider families of hash functions \mathcal{H} that offer relaxed static guarantees – that is, our results still apply to families either make assumptions about the input set [25] or require the use of a small auxiliary stash [4, 19].

Unlike prior algorithms, the Dancing-Kickout Algorithm takes a *non-greedy* approach to record-eviction. The algorithm will sometimes continue a kickout chain *past* a bin that has a free slot, in order to avoid “hotspot clusters” of full bins within the hash table. These hotspots are avoided by ensuring that, whenever a bin surrenders its final free slot, the bin is at the end of a reasonably long random walk, and is thus itself a “reasonably” random bin. Intuitively, the random structure that the algorithm instills into the hash table makes it possible for the hash functions from \mathcal{H} to not be fully random.

The problem of low-latency Cuckoo hashing is closely related to the problem of incremental edge orientation. In particular, the static guarantee for a Cuckoo hash table (with bins of size one) means that the edges in a certain graph form a pseudoforest. And the problem of dynamically maintaining a Cuckoo hash table (with bins of size $O(1)$) can be solved by dynamically orienting the pseudoforest in order to maintain constant out-degrees. The Dancing-Kickout algorithm is derived by applying our results for incremental edge orientation along with several additional ideas to handle deletions.

In addition to maintaining n bins, the Dancing-Kickout Algorithm uses an auxiliary data structure of size $O(n)$. The data structure incurs at most $O(1)$ modifications per insertion/deletion. Importantly, the auxiliary data structure is not accessed during queries, which continue to be implemented as in a standard Cuckoo hash table.

Our results come with an interesting lesson regarding the symbiotic relationship between Cuckoo hashing and edge orientation. There has been a great deal of past work on Cuckoo hashing that focuses on parameters such as associativity, number of hash functions, and

choice of hash function. We show that a new dimension that also warrants attention: how to dynamically maintain the table to ensure that a short kickout chain exists for every insertion. Algorithms that greedily optimize *any given operation* (e.g., random walk and BFS) may inadvertently structure the table in a way that compromises the performance of some later operations. In contrast, the non-greedy approach explored in this paper is able to offer strong performance guarantees for all operations, even if the hash functions being used are far from fully random. The results in this paper apply only to 1-associative static guarantees, and are therefore innately limited in the types of dynamic guarantees that they can offer (for example, we cannot hope to support a load factor of better than 0.5). An appealing direction for future work is to design and analyze eviction algorithms that offer strong dynamic guarantees in hash tables with either a large associativity or a large number of hash functions – it would be especially interesting if such guarantees could be used to support a load factor of $1 - q$ for an arbitrarily small positive constant q .

Related work on low-latency hash tables

Several papers have used ideas from Cuckoo hashing as a parts of new data structures that achieve stronger guarantees. Arbritman et al. [2] showed how to achieve a fully constant-time hash table by maintaining a polylogarithmic-size backyard consisting of the elements whose insertions have not yet completed at any given moment. Subsequent work then showed that, by storing almost all elements in a balls-in-bins system and then storing only a few “overflow” elements in a backyard Cuckoo hash table, one can construct a succinct constant-time hash table [3].²

Whereas the focus of these papers [2, 3] is to design new data structures that build on top of Cuckoo hashing, the purpose of our results is to consider *standard* Cuckoo hashing but in the dynamic setting. In particular, our goal is to show that dynamic guarantees for Cuckoo hashing do not have to be restricted to fully random hash-functions; by using the Dancing-Kickout Algorithm for maintaining the Cuckoo hash table, *any* family of hash functions that enjoys static guarantees can also enjoy dynamic guarantees.

2 An Algorithm with High-Probability Worst-Case Guarantees

This section considers the problem of incremental edge orientation in a forest. Let e_1, \dots, e_{n-1} be a sequence of edges between vertices in $V = \{v_1, \dots, v_n\}$ such that the edges form a tree on the vertices.

Our algorithm, which we call the Dancing-Walk Algorithm, guarantees out-degree at most 3 for each vertex, and performs at most $O(\log \log n)$ edge-flips per operation. Each step of the algorithm takes time at most $O(\log n \log \log n)$ to process. The algorithm is randomized, and can sometimes declare failure. The main technical difficulty in analyzing the algorithm is to show that the probability of the algorithm declaring failure is always very small.

We now describe the Dancing-Walk Algorithm. At any given moment, the algorithm allows each vertex v to have up to two **primary** out-going edges, and one **secondary** out-going edge. A key idea in the design of the algorithm is that, once a vertex has two primary out-going edges, the vertex can **volunteer** to take on a secondary out-going edge in order to ensure that a chain of edge flips remains short. But if vertices volunteer too

² It is worth noting, however, that as discussed in [17], the data structure of [3] can be modified to use any constant-time hash table in place of deamortized Cuckoo hashing.

frequently in some part of the graph, then the supply of potential volunteers will dwindle, which would destroy the algorithm's performance. The key is to design the algorithm in a way so that volunteering vertices are able to be useful but are not overused.

Consider the arrival of a new edge e_i . Let v_1 and v_2 be the two vertices that e_i connects, and let T_1 and T_2 be the two trees rooted at v_1 and v_2 , respectively. The algorithm first determines which of T_1 or T_2 is smaller (for this description we will assume $|T_1| \leq |T_2|$). Note that, by maintaining a simple union-find data structure on the nodes, the algorithm can recover the sizes of T_1 and T_2 each in $O(\log n)$ time.

The algorithm then performs a random walk through the (primary) directed edges of T_1 , beginning at v_1 (we call v_1 the **source vertex** of the random walk). Each step of the random walk travels to a new vertex by going down a random outgoing primary edge from the current vertex. If the random walk encounters a vertex u with out-degree less than 2 (note that this vertex u may even be v_1), then the walk terminates at that vertex. Otherwise, the random walk continues for a total of $c \log \log n$ steps, terminating at some vertex u with out-degree either 2 or 3. If the final vertex u has out-degree 2, meaning that the vertex does not yet have a secondary out-going edge, then the vertex u volunteers to take a secondary out-going edge and have its out-degree incremented to 3. If, on the other hand, the final vertex u already has out-degree 3, then the random walk is considered to have **failed**, and the random walk is repeatedly restarted from scratch until it succeeds. The algorithm performs up to $d \log n$ random-walk attempts for some sufficiently large constant d ; if all of these fail, then the algorithm **declares failure**.

Once a successful random walk is performed, all of the edges that the random walk traveled down to get from v_1 to u are flipped. This decrements the degree of v_1 and increments the degree of u . The edge e_i is then oriented to be out-going from v_1 . The result is that every vertex in the graph except for u has unchanged out-degree, and that u has its out-degree incremented by 1.

In the rest of the section, we prove the following theorem:

► **Theorem 1.** *With high probability in n , the Dancing-Walk Algorithm can process all of e_1, \dots, e_{n-1} without declaring failure. If the algorithm does not declare failure, then each step flips $O(\log \log n)$ edges and takes $O(\log n \log \log n)$ time. Additionally, no vertex's out-degree ever exceeds 3.*

For each edge e_t , let B_t be the binary tree in which the random walks are performed during the operation in which e_t is inserted. In particular, for each internal node of B_t , its children are the vertices reachable by primary out-going edges; all of the leaves in B_t are either at depth $c \log \log n$, or are at smaller depth and correspond with a vertex that has out-degree one or zero. Note that the set of nodes that make up B_t is a function of the random decisions made by the algorithm in previous steps, since these decisions determine the orientations of edges. Call the leaves at depth $(c \log \log n)$ in B_t the **potential volunteer leaves**. If every leaf in B_t is a potential volunteer leaf, then B_t can have as many as $(\log n)^c$ such leaves.

The key to proving Theorem 1 is to show with high probability in n , that for each step t , the number of potential volunteer leaves in B_t that have already volunteered in previous steps is at most $(\log n)^c / 2$.

► **Proposition 2.** *Consider a step $t \in \{1, 2, \dots, n-1\}$. With high probability in n , the number of potential volunteer leaves in B_t that have already volunteered in previous steps is at most $(\log n)^c / 2$.*

Assuming the high-probability outcome in Proposition 2, it follows that each random walk performed during the t -th operation has at least a $1/2$ chance of success. In particular, the only way that a random walk can fail is if it terminates at a leaf of depth $c \log \log n$ and that leaf has already volunteered in the past. With high probability in n , one of the first $O(\log n)$ random-walk attempts will succeed, preventing the algorithm from declaring failure.

The intuition behind Proposition 2 stems from two observations:

- **The Load Balancing Property:** Each vertex v is contained in at most $\log n$ trees B_t . This is because, whenever two trees T_1 and T_2 are joined by an edge e_t , the tree B_t is defined to be in the smaller of T_1 or T_2 . In other words, for each step t that a vertex v appears in B_t , the size of the (undirected) tree containing v at least doubles.
- **The Sparsity Property:** During a step t , each potential volunteer leaf in B_t has probability at most $\frac{d \log n}{\log^c n}$ of being selected to volunteer.

Assuming that most steps succeed within the first few random-walk attempts, the two observations combine to imply that most vertices v are never selected to volunteer.

The key technical difficulty comes from the fact that the structure of the tree B_t , as well as the set of vertices that make up the tree, is partially a function of the random decisions made by the algorithm in previous steps. This means that the set of vertices in tree B_t can be partially determined by which vertices have or have not volunteered so far. In this worst case, this might result in B_t consisting entirely of volunteered vertices, despite the fact that the vast majority of vertices in the graph have not volunteered yet.

How much flexibility is there in the structure of B_t ? One constraint on B_t is that it must form a subtree of the undirected graph $G_t = \{e_1, \dots, e_{t-1}\}$. This constraint alone is not very useful. For example, if G_t is a $(\log^{c+1} n)$ -ary tree of depth $c \log \log n$, and if each node in G_t has volunteered previously with probability $1/\log^c n$, then there is a reasonably high probability that every internal node of G_t contains at least two children that have already volunteered. Thus there would exist a binary subtree of G_t consisting entirely of nodes that have already volunteered.

An important property of the Dancing-Walk Algorithm is that the tree B_t cannot, in general, form an arbitrary subtree of G_t . Lemma 3 bounds the number of possibilities for B_t :

► **Lemma 3.** *For a given sequence of edge arrivals e_1, \dots, e_{n-1} , the number of possibilities for tree B_t is at most $(\log n)^{2 \log^c n}$.*

Proof. We will show that, for a given node v in B_t , there are only $\log n$ options for who each of v 's children can be in B_t . In other words, B_t is a binary sub-tree of a $(\log n)$ -ary tree with depth $c \log \log n$. Once this is shown, the lemma can be proven as follows. One can construct all of the possibilities for B_t by beginning with the root node v_1 and iteratively by adding one node at a time from the top down. Whenever a node v is added, and is at depth less than $c \log \log n$, one gets to either decide that the node is a leaf, or to select two children for the node. It follows that for each such node v there are at most $\binom{\log n}{2} + 1 \leq \log^2 n$ options for what v 's set of children looks like. Because B_t can contain at most $\log^c n - 1$ nodes v with depths less than $c \log \log n$, the total number of options for B_t is at most $(\log^2 n)^{\log^c n}$, as stated by the lemma.

It remains to bound the number of viable children for each node v in B_t . To do this, we require a stronger version of the load balancing property. The Strong Load Balancing Property says that, not only is the number of trees B_t that contain v bounded by $\log n$, but the set of $\log n$ trees B_t that can contain v is a function only of the edge sequence (e_1, \dots, e_{n-1}) , and not of the randomness in the algorithm.

- **The Strong Load Balancing Property:** For each vertex v , there is a set $S_v \subseteq [n]$ determined by the edge-sequence (e_1, \dots, e_{n-1}) such that: (1) the set's size satisfies $|S_v| \leq \log n$, and (2) every B_t containing v satisfies $t \in S_v$.

The Strong Load Balancing Property is a consequence of the fact that, whenever a new edge e_t combines two trees T_1 and T_2 , the algorithm focuses only on the smaller of the two trees. It follows that a vertex v can only be contained in tree B_t if the size of the (undirected) tree containing v at least doubles during the t -th step of the algorithm. For each vertex v , there can only be $\log n$ steps t in which the tree size containing v doubles, which implies the Strong Load Balancing Property.

Consider a step t , and suppose that step t orients some edge e to be facing out from some vertex v . Then it must be that the path from edge e_t to vertex v goes through e as its final edge. In other words, for a given step t and a given vertex v , there is only one possible edge e that might be reoriented during step t to be facing out from v . By the Strong Load Balancing Property, it follows that for a given vertex v , there are only $\log n$ possibilities for out-going edges e . This completes the proof of the lemma. ◀

Now that we have a bound on the number of options for B_t , the next challenge is to bound the probability that a given option for B_t has an unacceptably large number of volunteered leaves.

The next lemma proves a concentration bound on the number of volunteered vertices in a given set. Note that the event of volunteering is not independent between vertices. For example, if two vertices v and u are potential volunteer leaves during some step, then only one of v or u can be selected to volunteer during that step.

► **Lemma 4.** *Fix a sequence of edge arrivals e_1, \dots, e_{n-1} , and a set S of vertices. The probability that every vertex in S volunteers by the end of the algorithm is at most, $O\left(n/(\log^{(c-3)}|S|)\right)$.*

Proof. For each step $t \in \{1, 2, \dots, n-1\}$, define F_t to be the number of elements of S that are potential volunteer leaves during step t . Define $p_t = \frac{F_t \cdot d \log n}{\log^c n}$, where $d \log n$ is the number of random-walk attempts that the algorithm is able to perform in each step before declaring failure. By the Sparsity Property, the value p_t is an upper bound for the probability that any of the elements of S volunteer during step t . In other words, at the beginning of step t , before any random-walk attempts are performed, the probability that some element of S volunteers during step t is at most p_t .

Note that the values of p_1, \dots, p_{n-1} are not known at the beginning of the algorithm. Instead, the value of p_t is partially a function of the random decisions made by the algorithm in steps $1, 2, \dots, t-1$. The sum $\sum_t p_t$ is deterministically bounded, however. In particular, since each vertex $s \in S$ can appear as a potential volunteer leaf in at most $\log n$ steps (by the Load Balancing Property), the vertex s can contribute at most $d \log^2 n$ to the sum $\sum_t p_t$. It follows that $\sum_t p_t \leq \frac{|S| d \log^2 n}{\log^c n}$.

Let X_t be the indicator random variable for the event that some vertex in S volunteers during step t . Each X_t occurs with probability at most p_t . The events X_t are not independent, however, since the value of p_t is not known until the end of step $t-1$. Nonetheless, the fact that $\sum_t p_t$ is bounded allows for us to prove a concentration bound on $\sum_t X_t$ using the following version of Azuma's inequality [22].

▷ **Claim 5.** Let $\mu \in [0, n]$, and suppose that Alice is allowed to select a sequence of numbers p_1, p_2, \dots, p_k , $p_i \in [0, 1]$, such that $\sum_i p_i \leq \mu$. Each time Alice selects a number p_i , she wins 1 dollar with probability p_i . Alice is an adaptive adversary in that she can take into account

12:10 Incremental Edge Orientation in Forests

the results of the first i bets when deciding on p_{i+1} . If X is Alice's profit from the game,

$$\Pr \left[X > (1 + \delta)\mu \right] \leq \exp((\delta - \ln(1 + \delta))(1 + \delta)\mu),$$

for all $\delta > 0$.

Applying Claim 5 to $X = \sum_t X_t$, with $\delta = \frac{\log^c n}{d \log^2 n} - 1$ and $\mu = \frac{|S| d \log^2 n}{\log^c n}$ (so that $(\delta + 1)\mu = |S|$), we get that $\Pr[X > |S|]$ is at most

$$\exp \left(|S| - |S| \ln \frac{\log^c n}{d \log^2 n} \right) = O \left(\exp(-|S| \ln \log^{c-3} n) \right) = O \left(\log^{-(c-3)|S|} n \right). \quad \blacktriangleleft$$

Combining Lemmas 3 and 4, we can now prove Proposition 2.

Proof of Proposition 2. Consider a tree B_t . By Lemma 3, the number of options for B_t , depending on the behavior of the algorithm in steps $1, 2, \dots, t-1$, is at most $(\log n)^{2 \log^c n}$. For a given choice of B_t , there are at most $\binom{\log^c n}{\frac{1}{2} \log^c n} \leq 2^{\log^c n}$ ways to choose a subset S consisting of $\frac{\log^c n}{2}$ of the potential volunteer leaves. For each such set of leaves S , Lemma 4 bounds the probability that all of the leaves in S have already volunteered by,

$$O \left(\log^{-(c-3)|S|} n \right) = O \left(\log^{-(\log^c n)(c-3)/2} n \right).$$

Summing this probability over all such subsets S of all possibilities for B_t , the probability that B_t contains $\frac{\log^c n}{2}$ already-volunteered leaves is at most,

$$O \left((\log n)^{2 \log^c n} \cdot 2^{\log^c n} \cdot \log^{-(\log^c n)(c-3)/2} n \right) = O \left(\frac{(2 \log n)^{2 \log^c n}}{\log^{(\log^c n)(c-3)/2} n} \right).$$

For a sufficiently large constant c , this is at most $\frac{1}{n^{\omega(1)}}$. The proposition follows by taking a union bound over all $t \in \{1, 2, \dots, n-1\}$. \blacktriangleleft

We conclude the section with a proof of Theorem 1

Proof of Theorem 1. Consider a step t in which the number of potential volunteer leaves in B_t that have already volunteered is at most $\frac{1}{2} \log^c n$. The only way that a random walk in step t can fail is if it lasts for $c \log \log n$ steps (without hitting a vertex with out-degree 1 or 0) and it finishes at a vertex that has already volunteered. It follows that, out of the $\log^c n$ possibilities for a $(c \log \log n)$ -step random walk, at most half of them can result in failure. Since each random-walk attempt succeeds with probability at least $1/2$, and since the algorithm performs up to $d \log n$ attempts for a large constant d , the probability that the algorithm fails on step t is at most $\frac{1}{n^d} = \frac{1}{\text{poly} n}$.

The above paragraph establishes that, whenever the search tree B_t contains at most $\frac{1}{2} \log^c n$ potential volunteer leaves that have already volunteered, then step t will succeed with high probability in n . It follows by Proposition 2 that every step succeeds with high probability in n .

We complete the theorem by discussing the properties of the algorithm in the event that it does not declare failure. Each step flips at most $O(\log \log n)$ edges and maintains maximum out-degrees of 3. Because each step performs at most $O(\log n)$ random-walk attempts, these attempts take time at most $O(\log n \log \log n)$ in each step. Additionally, a union-find data structure is used in order to allow for the sizes $|T_1|$ and $|T_2|$ of the two trees being combined to be efficiently computed in each step. Because the union-find data structure can be implemented to have worst-case operation time $O(\log n)$, the running time of each edge-insertion remains at most $O(\log n \log \log n)$. \blacktriangleleft

The tradeoff between edges flipped and out-degree

To conclude the section, we consider a modification of the Dancing-Walk Algorithm in which nodes are permitted to have out-degree as large as $k = \log^\varepsilon n + 1$ (instead of 3) for some parameter ε . Rather flipping the edges in a random walk of length $c \log \log n$, the new algorithm instead flips the edges in a random walk of length $c\varepsilon^{-1}$. The length of the random walk is parameterized so that the number of potential volunteers $|D_t|$ is still $\log^c n$, as was the case before. The resulting algorithm yields the following result, which allows for a tradeoff between edges flipped per insertion and maximum out-degree:

► **Theorem 6.** *Consider the Dancing-Walk Algorithm with maximum out-degree $k + 1$. With high probability in n , the algorithm can process all of e_1, \dots, e_{n-1} without declaring failure. If the algorithm does not declare failure, then each step flips $O(\log_k \log n)$ edges and takes $O(\log n \log_k \log n)$ time. Additionally, no vertex's out-degree ever exceeds $k + 1$.*

The proof of Theorem 6 can be found in the extended version of the paper [6].

3 Achieving Constant Amortized Running Time

We now discuss how to modify the Dancing-Walk Algorithm to achieve a total running time of $X = O(n)$. The resulting algorithm, which we call the **Rank-Based Dancing-Walk Algorithm** offers the following guarantee on top of those in Theorem 1:

► **Theorem 7.** *To perform $n - 1$ edge insertions, the total time required by the Rank-Based Dancing-Walk Algorithm is at most $O(n)$ with high probability in n .*

To simplify the discussion in this section, we focus here on the simpler problem of bounding the *expected* total running time $\mathbb{E}[X]$. The full proof of Theorem 7 can be found in the extended version of the paper [6].

Although each random walk is permitted to have length as large as $\Theta(\log \log n)$, one can easily prove that a random walk through a tree of m nodes expects to hit a node with out-degree less than 2 within $O(\log m)$ steps. Recall that, whenever an edge e_t combines two (undirected) trees T_1 and T_2 , the ensuing random walks are performed in the *smaller* of T_1 or T_2 . The expected contribution to the running time X is therefore, $O(\min(\log |T_1|, \log |T_2|))$. That is, even though a given edge-insertion e_t could incur up to $\Theta(\log n)$ random walks each of length $\Theta(\log \log n)$ in the worst case, the expected time spent performing random walks is no more than $O(\min(\log |T_1|, \log |T_2|))$.

Let \mathcal{T} denote the set of pairs (T_1, T_2) that are combined by each of the $n - 1$ edge insertions. A simple amortized analysis shows that

$$\sum_{(T_1, T_2) \in \mathcal{T}} \min(\log |T_1|, \log |T_2|) = O(n). \quad (1)$$

Thus the time spent performing random walks is $O(n)$ in expectation.

In addition to performing random walks, however, the algorithm must also compare $|T_1|$ and $|T_2|$ on each edge insertion. But maintaining a union-find data structure to store the sizes of the trees requires $\Omega(\alpha(n, n))$ amortized time per operation [16], where $\alpha(n, n)$ is the inverse Ackermann function.

Thus, for the algorithm described so far, the maintenance of a union-find data structure prevents an amortized constant running time per operation. We now describe how to modify the algorithm in order to remove this bottleneck.

Replacing size with combination rank

We modify the Dancing-Walk Algorithm so that the algorithm no longer needs to keep track of the size $|T|$ of each tree in the graph. Instead the algorithm keeps track of the **combination rank** $R(T)$ of each tree T – whenever two trees T_1 and T_2 are combined by an edge insertion, the new tree T_3 has combination rank,

$$R(T_3) = \begin{cases} \max(R(T_1), R(T_2)) & \text{if } R(T_1) \neq R(T_2) \\ R(T_1) + 1 & \text{if } R(T_1) = R(T_2). \end{cases}$$

Define the **Rank-Based Dancing-Walk Algorithm** to be the same as the Dancing-Walk Algorithm, except that the source vertex (i.e., the vertex from which the random walk begins) is selected to be in whichever of T_1 or T_2 has smaller combination rank (rather than smaller size).

The advantage of combination rank is that it can be efficiently maintained using a simple tree structure. Using this data structure, the time to merge two trees T_1 and T_2 (running the Dancing-Walk Algorithm with appropriately chosen source vertex) becomes simply $\min(R(T_1), R(T_2))$. This, in turn, can be upperbounded by $O(\min(\log |T_1|, \log |T_2|))$. By (1), the total time spent maintaining combination ranks of trees is $O(n)$.

The other important feature of combination rank is that it preserves the properties of the algorithm that are used to analyze correctness. Importantly, whenever a tree T is used for path augmentation by an edge-insertion e_t , the combination rank of T increases due to that edge insertion. One can further prove that the combination rank never exceeds $O(\log n)$, which allows one to derive the Strong Load Balancing Property and the Preset Children Property.

The disadvantage: longer random walks

The downside of using combination rank to select trees is that *random walks* can now form a running-time bottleneck. Whereas the expected running time of all random walks was previously bounded by (1), we now claim that it is bounded by,

$$\sum_{(T_1, T_2) \in \mathcal{T}} \left(\begin{cases} \log |T_1| & \text{if } R(T_1) \leq R(T_2) \\ \log |T_2| & \text{if } R(T_2) < R(T_1) \end{cases} \right) = O(n). \quad (2)$$

We now justify this claim. The problem is that a tree T can potentially have very small combination rank (e.g., $O(1)$) but very large size (e.g., $\Omega(n)$). As a result, the summation (1) may differ substantially from the summation (2).

Rather than bounding (2) directly, we instead examine the smaller quantity,

$$\sum_{(T_1, T_2) \in \mathcal{T}} \left(\begin{cases} \log |T_1| - R(T_1) & \text{if } R(T_1) \leq R(T_2) \\ \log |T_2| - R(T_2) & \text{if } R(T_2) < R(T_1) \end{cases} \right) = O(n). \quad (3)$$

The difference between (2) and (3) is simply

$$\sum_{(T_1, T_2) \in \mathcal{T}} \min(R(T_1), R(T_2)) = O(n),$$

meaning that an upper bound on (3) immediately implies an upper bound on (2).

The key feature of (3), however, is that it yields to a simple potential-function based analysis. In particular, if we treat each vertex v as initially having $\Theta(1)$ tokens, and we treat each tree combination (T_1, T_2) as incurring a cost given by the summand in (3), then one

can show that every tree T always has at least $\Omega\left(\frac{|T|}{2^{R(T)}}\right)$ tokens, which means that the total number of tokens spent is $O(n)$. This allows us to bound (3) by $O(n)$, which then bounds (2) by $O(n)$, and implies a total expected running time of $\mathbb{E}[X] = O(n)$.

4 Dynamic Cuckoo Hashing

In this section we present the *Dancing-Kickout Algorithm* for maintaining a Cuckoo hash table. For any family of hash functions \mathcal{H} that provides a 1-associative static guarantee, the Dancing-Kickout Algorithm offers a $O(1)$ -associative dynamic guarantee using the same hash-function family \mathcal{H} .

We will state our results so that they also apply to Cuckoo hashing with a stash [4, 19]. A Cuckoo hash table with a **stash** of size s is permitted to store s elements *outside* of the table in a separate list. Having a small stash has been shown by past work to significantly simplify the problem of achieving high-probability static guarantees [4] – our results can be used to make these guarantees dynamic.

Let $h = (h_1, h_2)$ be a pair of hash functions mapping records to $[n]$. A set X of records is ***h-viable*** if it is possible to place the records X into a 1-associative n -bin Cuckoo hash table using hash functions h_1 and h_2 .

Even if a set of records X is not h -viable, it may be that there is a set of s elements Y for which $X \setminus Y$ is h -viable. In this case, we say X is ***h-viable with a stash of size s***.

Past static guarantees [1, 4, 5, 15, 25, 27, 30] for a hash family \mathcal{H} , have taken the following form, where $c \in (0, 1)$, $p(n) \in \text{poly}(n)$, $s \in O(1)$ are parameters: Every set of records X of size cn has probability at least $1 - 1/p(n)$ of being h -viable with a stash of size s , where $h = (h_1, h_2)$ is drawn from \mathcal{H} . In addition to considering guarantees of this type, a fruitful line of work [25] has also placed additional restrictions on the set X of records (namely, that X exhibits high entropy). In this section, we will state our results in such a way so that they are applicable to all of the past variants of static guarantees that we are aware of.

Viability as a graph property

Define the *Cuckoo graph* $G(X, h)$ for a set of records X and for a pair of hash functions $h = (h_1, h_2)$ to be the graph with vertices $[n]$ and with (undirected) edges $\{(h_1(x), h_2(x)) \mid x \in X\}$. The problem of configuring where records should go in the hash table corresponds to an edge-orientation problem in G . In particular, one can think of each record x that resides in a bin $h_i(x)$ as representing an edge $(h_1(x), h_2(x))$ that is oriented to face out of vertex $h_i(x)$. A set of records X is h -viable if and only if the edges in $G(X, h)$ can be oriented so that the maximum out-degree is 1.

Similarly, a set of records X is h -viable with a stash of size s if and only if there are s (or fewer) edges that can be removed from the Cuckoo graph $G(X, h)$ so that the new graph G' can be oriented to have maximum out-degree 1.

Applying static guarantees to dynamic settings

In order to apply static guarantees in a dynamic setting, we define the notion of a sequence of insert/delete operations satisfying a static guarantee.

For $\varepsilon \in (0, 1)$ and for a hash-function pair $h = (h_1, h_2)$, we say that a sequence $\Psi = \langle \psi_1, \psi_2, \dots \rangle$ of insert/delete operations is ***(ε, h)-viable with a stash of size s*** if the following holds: for every subsequence of operations of the form $P_i = \langle \psi_{i\varepsilon n+1}, \psi_{i\varepsilon n+2}, \dots, \psi_{(i+1)\varepsilon n} \rangle$, the set X of records that are present (at any point) during the operations P_i has the property that X is h -viable with a stash of size s .

The dynamic guarantees in this section will assume only that the sequence of operations Ψ is (ε, h) -viable (with a stash of size s) for some known parameter $\varepsilon \in (0, 1)$, and will make no other assumptions about Ψ or the hash-function pair $h = (h_1, h_2)$.

Note that the property of being (ε, h) -viable is a statement about the sets of records X that are present during windows of εn operations. If the table is always filled to capacity cn , for some $c \in (0, 1)$, then the property of being (ε, h) -viable is a statement about sets of $(c + \varepsilon)n$ records. Thus dynamic guarantees for tables on cn records can be derived from static guarantees that apply to tables of $(c + \varepsilon)n$ records. By making ε smaller, one can close the gap between the capacities for the static and dynamic guarantees – but as we shall see, this also increases the constant in the algorithm’s running time.

Our dynamic guarantee

Formally, we say that an *implementation of a k -associative Cuckoo hash table with a stash of size s* is an algorithm that maintains a Cuckoo hash table with n bins, each of size k , and with a stash of size up to s . The implementation is given two hash functions h_1, h_2 , and every record x in the table must either be stored in one of the bins $h_1(x), h_2(x)$ or in the stash. The implementation is permitted to maintain an additional $O(n)$ -space data structure \mathcal{D} for additional bookkeeping, as long as \mathcal{D} is not modified by queries, and as long as each insert/delete incurs at most $O(1)$ writes to \mathcal{D} .

We say that a Cuckoo hash table implementation satisfies *the dynamic guarantee* on a sequence of operations Ψ , if:

- Each insert/delete operation incurs $O(\log \log n)$ kickouts and takes time $O(\log n \log \log n)$.
- The amortized cost of each insert/delete operation is $O(1)$.

The goal of this section will be to describe an implementation of Cuckoo hashing that offers the dynamic guarantee (with high probability) as long as the underlying sequence of operations Ψ is (ε, h) -viable. We call our implementation of Cuckoo hashing the **Dancing-Kickout Algorithm**.

► **Theorem 8.** *Let $\varepsilon \in (0, 1)$ and s be constants (s may be 0). Let $h = (h_1, h_2)$ be a pair of hash functions. Let Ψ be a sequence of $\text{poly}(n)$ insert/delete operations that is (ε, h) -viable with a stash of size s .*

With high probability in n , the Dancing-Kickout Algorithm implements an 8-associative Cuckoo hash table with a stash of size s that satisfies the dynamic guarantee on Ψ .

Proof. We take the approach of starting with a weaker version of the theorem and then working our way towards the full version. Initially we will consider only inserts, but no deletes or stash. Then we will consider only inserts and a stash, but no deletes. Then we will consider all of inserts, deletes, and a stash, but we will make what we call the **full-viability assumption**, which is that the set X of *all* of records inserted and deleted by Ψ is h -viable. Finally, we will show how to remove the full-viability assumption.

We begin by describing the Dancing-Kickout Algorithm in the case where Ψ consists of only insertions (and no deletions). In this case, the algorithm only uses the first 4 slots in each bin. We also begin with the simplifying assumption that the stash size s is 0.

The algorithm thinks of each record x as representing an edge $(h_1(x), h_2(x))$ in the Cuckoo graph G . Since the set of records X being inserted is h -viable, it must be that G can be oriented with out-degree 1. This means that each connected component in G is a pseudotree (i.e., a tree with up to one additional edge added).

In this case, the Dancing-Kickout Algorithm works as follows. Whenever an edge-insertion connects two vertices from different connected components, the Dancing-Kickout Algorithm simply uses the Rank-Based Dancing-Walk Algorithm to maintain an edge-orientation with maximum out-degree 3. On the other hand, when an edge-insertion connects two vertices v, u that are already in the same tree as one another (we call the edge (v, u) a **bad edge**), the Dancing-Kickout Algorithm orients the edge arbitrarily and then disregards that edge in all steps (i.e., the edge cannot be used as part of a random walk). Since G is a pseudoforest, each vertex v is incident to at most one bad edge; it follows that the maximum out-degree in the graph never exceeds 4. This, in turn, means that no bin in the Cuckoo hash table stores more than 4 items.

The analysis of the Rank-Based Dancing-Walk algorithm ensures that the edge-insertions involving good edges satisfy the dynamic guarantee with high probability in n (that is, each operation takes time $O(\log n \log \log n)$, incurs $O(\log \log n)$ edge flips, and takes amortized time $O(1)$). The edge-insertions involving bad edges can be analyzed as follows. Note that the time for the Rank-Based Dancing-Walk Algorithm to identify that an edge $e = (v, u)$ is bad is just the height of the rank tree containing v and u . Since combination ranks never exceed $O(\log n)$, the time to identify a bad edge is never more than $O(\log n)$. Since each rank-tree will have at most one bad edge identified in it (because each connected component contains at most 1 bad edge), the total time spent identifying bad edges is at most the sum of the depths of the rank trees (at the end of all edge insertions); this, in turn, is $O(n)$ since the depth of each rank tree is never more than the number of elements it contains. Thus the operations in which bad edges are inserted do not cause the dynamic guarantee to be broken.

Now we describe what happens if Ψ still consists only of insertions, but a stash of size $s > 0$ is used. In this case, the Dancing-Kickout Algorithm places an edge $e = (v, u)$ in the stash (i.e., the algorithm places the record x for which $h_1(x) = v$ and $h_2(x) = u$ in the stash) if e is a bad edge *and* if both of the vertices v and u are already incident to bad edges. On the other hand, if one of v or u is not already incident to a bad edge, then the edge can be oriented out-going from that vertex (just as was the case without a stash). Call an edge e **super bad** if, when e is inserted, there is already a bad edge in the connected component containing e . Since Ψ is h -viable with a stash of size s , the number of super bad edges is at most s .³ Because the Random-Walk Algorithm only stashes super bad edges, the algorithm is guaranteed to never stash more than s records at a time. The running time of the algorithm on non-super-bad edges is the same as in the case of no stash; on the other hand, the s super bad edges can contribute $s \cdot O(\log n) = O(\log n)$ in total to the running time of the algorithm. Thus, with high probability, the Random-Walk Algorithm still satisfies the dynamic guarantee.

Now we consider what happens if Ψ contains deletes in addition to inserts. To begin, consider the special case where the set X of *all records* that Ψ *ever* inserts (including those that are subsequently deleted) has the property that X is h -viable – we call this the **full-viability assumption**. Under the full-viability assumption, deletes can be implemented with **tombstones**, meaning that when a record is deleted it is simply marked as deleted without actually being removed. In fact, the use of tombstones is not actually necessary. This is because the analysis of the Rank-Based Dancing-Walk Algorithm for edge-orientation

³ To see this formally, note that there must be a set of at most s edges Y such that $X \setminus Y$ is a pseudoforest. That is, without the edges Y there would be *no* super bad edges. On the other hand, one can verify that placing each of the edges from Y back into the sequence of edges $X \setminus Y$ adds at most $|Y|$ super bad edges, since each edge that is placed in can increase the number of super bad edges by at most 1.

continues to work without modification even if edges in the graph disappear arbitrarily over time, as long as all of the edges (*including those that disappear*) form a forest. Thus, in the case where the full-viability assumption holds, we can simply implement deletes by removing the appropriate record from the table, and then we can use the Dancing-Kickout algorithm exactly as described so far. Since the Rank-Based Dancing-Walk Algorithm can handle edges disappearing, it follows that the Dancing-Kickout algorithm still satisfies the dynamic guarantee with high probability.

Finally, we consider what happens if Ψ contains both inserts and deletes, but without making the full-viability assumption. So far, we have only used the first 4 slots of each bin. We now incorporate into the algorithm slots 5, 6, 7, 8, and we modify the algorithm to gradually rebuild the table in phases, where consecutive phases toggle between using only slots 1, 2, 3, 4 or using only slots 5, 6, 7, 8; as we shall see, each phase is designed so that the running-time of its operations can be treated as meeting the full-viability assumption.

In more detail, the algorithm performs gradual rebuilds as follows. The operations Ψ are broken into phases P_1, P_2, \dots each consisting of εn operations. At the beginning of each phase P_i where i is even (resp. i is odd), the hash table uses only the slots 1, 2, 3, 4 (resp. 5, 6, 7, 8) in each bin. During the phase of operations P_i , any new insertions are performed with the Dancing-Kickout Algorithm using slots 5, 6, 7, 8 (resp. 1, 2, 3, 4). Also, during the j -th operation in the phase P_i , the algorithm looks at bin j , takes any records in slots 1, 2, 3, 4 (resp. 5, 6, 7, 8), and reinserts those records into the hash table using slots 5, 6, 7, 8 (resp. 1, 2, 3, 4).⁴ Finally, deletes are implemented simply by removing the appropriate record x , regardless of what slot that record may be in.

During a given phase P_i , the algorithm can be thought of as starting with a new empty Cuckoo hash table (consisting in each bin of either the slots 1, 2, 3, 4 if i is odd or 5, 6, 7, 8 if i is even). Then over the course of P_i , one can think of the algorithm as performing not only the operations in P_i , but also populating the new hash table with any elements that were present at the beginning of the phase P_i (unless those elements are deleted before they have a chance to be re-populated). Let X be the set of all records x that are placed into the new hash table at some point during P_i (this includes both elements that operations in P_i act on, as well as elements that are re-inserted due to the gradual rebuild during the phase). By the (ε, h) -viability of Ψ , we know that X is h -viable. This means that phase P_i can be analyzed as satisfying the full-viability assumption. Thus, with high probability in n , the algorithm does not violate the dynamic guarantee during phase P_i . Since there are $\text{poly}(n)$ phases, it follows that, with high probability in n , the algorithm never violates the dynamic guarantee. ◀

References

- 1 Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, 2018.
- 2 Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *International Colloquium on Automata, Languages, and Programming*, pages 107–118. Springer, 2009.

⁴ Additionally, if a stash of size $s > 0$ is used, then the first operation of each phase P_i reinserts all of the elements in the stash, using only slots 5, 6, 7, 8 if i is even and only slots 1, 2, 3, 4 if i is odd.

- 3 Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 787–796. IEEE, 2010.
- 4 Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.
- 5 Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. A simple hash class with strong randomness properties in graphs and hypergraphs. *arXiv preprint arXiv:1611.00029*, 2016.
- 6 Michael A. Bender, Tsvi Kopelowitz, William Kuszmaul, Ely Porat, and Clifford Stein. Incremental edge orientation in forests. *arXiv preprint arXiv:2107.01250*, 2021.
- 7 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. *Algorithmica*, 82(2):245–259, 2020.
- 8 A. Bernstein and C. Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Part I*, pages 167–179, 2015.
- 9 A. Bernstein and C. Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 692–711, 2016.
- 10 G. Stølting Brodal and R. Fagerberg. Dynamic representation of sparse graphs. In *Algorithms and Data Structures, 6th International Workshop, WADS*, pages 342–351, 1999.
- 11 Jeffrey S Cohen and Daniel M Kane. Bounds on the independence required for cuckoo hashing. *ACM Transactions on Algorithms*, 2009.
- 12 Søren Dahlgaard and Mikkel Thorup. Approximately minwise independence with twisted tabulation. In *Scandinavian Workshop on Algorithm Theory*, pages 134–145. Springer, 2014.
- 13 Luc Devroye and Pat Morin. Cuckoo hashing: further analysis. *Information Processing Letters*, 86(4):215–219, 2003.
- 14 Martin Dietzfelbinger and Ulf Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 795–804. SIAM, 2009.
- 15 Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 629–638, 2003.
- 16 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 345–354, 1989.
- 17 Michael T Goodrich, Daniel S Hirschberg, Michael Mitzenmacher, and Justin Thaler. Fully de-amortized cuckoo hashing for cache-oblivious dictionaries and multimaps. *arXiv preprint arXiv:1107.4378*, 2011.
- 18 M. He, G. Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Algorithms and Computation - 25th International Symposium, ISAAC*, pages 128–140, 2014.
- 19 Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2010.
- 20 T. Kopelowitz, R. Krauthgamer, E. Porat, and S. Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP(2)*, pages 532–543, 2014.
- 21 L. Kowalik and M. Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms*, 2(3):335–363, 2006.
- 22 William Kuszmaul and Qi Qi. The multiplicative version of azuma’s inequality, with an application to contention analysis. *arXiv preprint arXiv:2102.05077*, 2021.
- 23 Xiaozhou Li, David G Andersen, Michael Kaminsky, and Michael J Freedman. Algorithmic improvements for fast concurrent cuckoo hashing. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14, 2014.

- 24 Michael Mitzenmacher. Some open questions related to cuckoo hashing. In *European Symposium on Algorithms*, pages 1–10. Springer, 2009.
- 25 Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 746–755. Society for Industrial and Applied Mathematics, 2008.
- 26 O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013. doi:10.1145/2488608.2488703.
- 27 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *European Symposium on Algorithms*, pages 121–133. Springer, 2001.
- 28 Rina Panigrahy. Efficient hashing with lookups in two memory accesses. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839, 2005.
- 29 Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 209–228. SIAM, 2013.
- 30 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM (JACM)*, 59(3):1–50, 2012.
- 31 Mikkel Thorup. Fast and powerful hashing using tabulation. *Communications of the ACM*, 60(7):94–101, 2017.

k -Center Clustering with Outliers in the Sliding-Window Model

Mark de Berg ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Morteza Monemizadeh ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Yu Zhong ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

The k -center problem for a point set P asks for a collection of k congruent balls (that is, balls of equal radius) that together cover all the points in P and whose radius is minimized. The k -center problem with outliers is defined similarly, except that z of the points in P do need not to be covered, for a given parameter z . We study the k -center problem with outliers in data streams in the sliding-window model. In this model we are given a possibly infinite stream $P = \langle p_1, p_2, p_3, \dots \rangle$ of points and a time window of length W , and we want to maintain a small sketch of the set $P(t)$ of points currently in the window such that using the sketch we can approximately solve the problem on $P(t)$.

We present the first algorithm for the k -center problem with outliers in the sliding-window model. The algorithm works for the case where the points come from a space of bounded doubling dimension and it maintains a set $S(t)$ such that an optimal solution on $S(t)$ gives a $(1 + \varepsilon)$ -approximate solution on $P(t)$. The algorithm uses $O((kz/\varepsilon^d) \log \sigma)$ storage, where d is the doubling dimension of the underlying space and σ is the spread of the points in the stream. Algorithms providing a $(1 + \varepsilon)$ -approximation were not even known in the setting without outliers or in the insertion-only setting with outliers. We also present a lower bound showing that any algorithm that provides a $(1 + \varepsilon)$ -approximation must use $\Omega((kz/\varepsilon) \log \sigma)$ storage.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Streaming algorithms, k -center problem, sliding window, bounded doubling dimension

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.13

Funding MdB is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

Clustering is one of the most important tools to analyze large data sets. A well-known class of clustering algorithms is formed by centroid-based algorithms, which include k -means clustering, k -median clustering and k -center clustering. The latter type of clustering is the topic of our paper. In the k -center problem one is given a set P of points from a metric space and a parameter k , and the goal is to find k congruent balls (that is, balls of equal radius) that together cover the points from P and whose radius is minimized. Note that the special case $k = 1$ corresponds to the minimum-enclosing ball problem. Data sets in practice often contain outliers, leading to the *k -center problem with outliers*. Here we are given, besides P and k , a parameter z that indicates the allowed number of outliers. Thus the radius of the balls in an optimal solution is given by

$\text{OPT}_{k,z}(P) :=$ the smallest radius ρ such that we can cover all points from P ,
except for at most z outliers, by k balls of radius ρ .



© Mark de Berg, Morteza Monemizadeh, and Yu Zhong;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 13; pp. 13:1–13:13

Leibniz International Proceedings in Informatics



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

In this paper we study the k -center problem with outliers in data streams, where the input is a possibly infinite stream $P = \langle p_1, p_2, \dots \rangle$ of points. The goal is to maintain a solution to the k -center problem as the points arrive over time, without any knowledge of future arrivals and using limited (sub-linear) storage. Since we cannot store all the points in the stream, we cannot expect to maintain an optimal solution. Hence, the two main quality criteria of a streaming algorithm are its approximation ratio and the amount of storage it uses. We will study this problem in the *sliding-window model*. In this model we are given a window length W and we are, at any time t , only interested in the points that arrived in the time window $(t - W, t]$. Working in the sliding-window model is often significantly more difficult than working in the standard (insertion-only) streaming model.

Previous work

Charikar et al. [6] were the first to study the metric k -center problem in data streams. They developed an algorithm in the insertion-only model that computes an 8-approximation for the k -center problem using $\Theta(k)$ space. Later McCutchen and Khuller [13] improved the approximation ratio to $2 + \varepsilon$ at the cost of increasing the storage to $O((k/\varepsilon) \log(1/\varepsilon))$. McCutchen and Khuller also studied the k -center problem with $z \geq 1$ outliers, for which they gave a $(4 + \varepsilon)$ -approximation algorithm that requires $O(kz/\varepsilon)$ space.

The above results are for general metric spaces. In spaces of *bounded doubling dimension*¹ better bounds are possible. Indeed, Ceccarello et al. [3] gave a $(3 + \varepsilon)$ -approximation algorithm for the k -center problem with z outliers, thus improving the approximation ratio $(4 + \varepsilon)$ for general metrics. Their algorithm requires $O((k + z)(1/\varepsilon)^d)$ storage, where d is the doubling dimension of the underlying space (which is assumed to be a fixed constant).

The algorithms mentioned so far are deterministic. Charikar et al. [8] and Ding et al. [10] studied sampling-based streaming algorithms for the Euclidean k -center problem with outliers, showing that if one allows slightly more than z outliers then randomization can reduce the storage requirements. Our focus, however, is on deterministic algorithms.

For the k -center problem in the sliding-window model, the only result we are aware of is due to Cohen-Addad et al. [9]. They deal with the k -center problem in general metric spaces, but without outliers, and they propose a $(6 + \varepsilon)$ -approximation algorithm using $O((k/\varepsilon) \log \sigma)$ storage, and a $(4 + \varepsilon)$ -approximation for the special case $k = 2$. Here σ denotes (an upper bound on) the spread of the points in the stream; in other words, $\sigma := \Delta_{\max}/\Delta_{\min}$, where Δ_{\max} is an upper bound on the maximum distance and Δ_{\min} is a lower bound on the minimum distance between any two distinct points. It is assumed that Δ_{\max} and Δ_{\min} are known to the algorithm. They also prove that any algorithm for the 2-center problem with outliers in general metric spaces that achieves an approximation ratio of less than 4 requires $\Omega(W^{1/3})$ space, where W is the size² of the window. Table 1 gives an overview of the known results on the k -center problem in the insertion-only and the sliding-window model.

While our main interest is in the k -center problem for $k > 1$, our results also apply when $k = 1$. Hence, we also briefly discuss previous results for the 1-center problem.

¹ The *doubling dimension* of a space X is the smallest number d such that any ball B in the space can be covered by 2^d balls of radius $\text{radius}(B)/2$.

² Here the window size W is defined in terms of the number of points in the window, that is, the window consists of the W most recent points. We define the window in a slightly more general manner, by defining W to be the length (that is, duration) of the window. Note that if we assume that the i -th point arrives at time $t = i$, then the two models are the same.

■ **Table 1** Results for the k -center problem with and without outliers in the insertion-only and the sliding-window model. Bounds on the storage are asymptotic. In the papers where the metric space has bounded doubling dimension or is Euclidean, the dimension d is considered a constant.

model	metric space	approx.	storage	outliers	ref.
insertion-only	general	8	k	no	[6]
	general	$2 + \varepsilon$	$(k/\varepsilon) \log(1/\varepsilon)$	no	[13]
	general	$4 + \varepsilon$	kz/ε	yes	[13]
	bounded doubling	$3 + \varepsilon$	$(k + z)/\varepsilon^d$	yes	[3]
sliding window	general	$6 + \varepsilon$	$(k/\varepsilon) \log \sigma$	no	[9]
	bounded doubling	$1 + \varepsilon$	$(kz/\varepsilon^d) \log \sigma$	yes	here

For the 1-center problem in d -dimensional Euclidean space, streaming algorithms that maintain an ε -kernel give a $(1 + \varepsilon)$ -approximation. An example is the algorithm of Zarrabi-Zadeh [14] which maintains an ε -kernel of size $O(1/\varepsilon^{(d-1)/2} \log(1/\varepsilon))$. Moreover, using only $O(d)$ storage one can obtain a 1.22-approximation for the 1-center problem without outliers [2, 4]. For the 1-center problem with outliers, one can obtain a $(1 + \varepsilon)$ -approximation algorithm that uses $z/\varepsilon^{O(d)}$ storage by the technique of Agarwal et al. [1]. Zarrabi-Zadeh and Mukhopadhyay [15] studied the 1-center problem with z outliers in high-dimensional Euclidean spaces, where d is not considered constant, giving a 1.73-approximation algorithm that requires $O(d^3 z)$ storage. Recently, Hatami and Zarrabi-Zadeh [12] extended this result to 2-center problem with z outliers, obtaining a $(1.8 + \varepsilon)$ -approximation using $O(d^3 z^2 + dz^4/\varepsilon)$ storage. None of the 1-center algorithms discussed above works in the sliding-window model.

A problem that is closely related to the 1-center problem is the *diameter problem*, where the goal is to maintain an approximation of the diameter of the points in the stream. This problem has been studied in the sliding-window model by Feigenbaum, Kannan, Zhang [11] and later by Chan and Sadjad [5], who gave a $(1 + \varepsilon)$ -approximation for the diameter problem (without outliers) in the sliding window model, using $O((1/\varepsilon)^{(d+1)/2} \log(\sigma/\varepsilon))$ storage.

Our contribution

We present the first algorithm for the k -center problem with z outliers in the sliding-window model. It works in spaces of bounded doubling dimension and yields a $(1 + \varepsilon)$ -approximation. So far a $(1 + \varepsilon)$ -approximation was not even known for the k -center problem without outliers in the insertion-only model. Our algorithm uses $O((kz/\varepsilon^d) \log \sigma)$ storage,³ where d is the doubling dimension and σ is the spread, as defined above. Thus for the 1-center problem we obtain a solution that uses $O((z/\varepsilon^d) \log \sigma)$ storage. This solution also works for the diameter problem. Note that also for the 1-center problem with outliers (and the diameter problem with outliers) an algorithm for the sliding-window model was not yet known. A useful property of the sketch⁴ maintained by our algorithm for the k -center problem, is that it can also be used for the k' -center problem for any $k' < k$, as well as for the diameter problem. As in the previous papers on the k -center problem (or the diameter problem) in the sliding-window model [5, 9, 11], we assume that Δ_{\max} and Δ_{\min} (upper and lower bounds on the maximum and minimum distance between any two points in the stream) are known to the algorithm.

³ To correctly state the bound for the case $z = 0$ as well, we should actually write $O((k(z + 1)/\varepsilon^d) \log \sigma)$. To keep the bound concise, we will just write $O((kz/\varepsilon^d) \log \sigma)$, however.

⁴ We use the word *sketch* even though we do not study how to compose the sketches for two separate streams into a sketch for the concatenation of the stream, as the term *sketch* seems more appropriate than *data structure*, for example.

As mentioned above, our algorithm provides a $(1 + \varepsilon)$ -approximation. More precisely, it maintains a set $S(t) \subset P(t)$ such that computing an optimal solution for $S(t)$ and then suitably expanding the balls in the solution, gives a $(1 + \varepsilon)$ -approximate solution for $P(t)$. Thus, to obtain a $(1 + \varepsilon)$ -approximate solution for $P(t)$ one needs to compute an optimal (or $(1 + \varepsilon')$ -approximate, for a suitable ε') solution for $S(t)$. Depending on the underlying space this can be slow, and it may be preferable to compute, say, a 2-approximation for $S(t)$, which would then give a $(2 + \varepsilon)$ -approximation for $P(t)$. The various options in Euclidean spaces and general spaces of bounded doubling dimension are discussed in Section 2.3.

Our second contribution is a lower bound for the k -center problem with outliers in the sliding-window model. Our lower bound shows that any algorithm that provides a $(1 + \varepsilon)$ -approximation must use $\Omega((kz/\varepsilon) \log \sigma)$ storage. This matches our upper bound up to the dependency on ε and d . The lower-bound construction uses points in \mathbb{R}^1 , so it shows that our algorithm is optimal in this case. The lower bound model is very general. It allows the algorithm to store points, or weighted points, or balls, or whatever it wants so that it can approximate an optimal solution for $P(t)$ at any time t . The main condition is that each object comes with an expiration time which corresponds to the expiration time of some point in the stream, and that the algorithm can only change its state when an object expires or a new object arrives; see Section 3 for details.

2 The algorithm

Let $P := \langle p_1, p_2, \dots \rangle$ be a possibly infinite stream of points from a metric space X of doubling dimension d and spread σ , where d is considered to be a fixed constant. We denote the arrival time of a point p_i by $t_{\text{arr}}(p_i)$. We say that p_i *expires* at time $t_{\text{exp}}(p_i) := t_{\text{arr}}(p_i) + W$, where W is the given length of the time window. To simplify the exposition, we assume that all arrival times and departure times (that is, times at which a point expires) are distinct. For a time t we define $P(t)$ to be the set⁵ of points currently in the window. In other words, $P(t) := \{p_i : t_{\text{arr}}(p_i) \leq t < t_{\text{exp}}(p_i)\}$. For a point $q \in X$ and a parameter $r \geq 0$, we use $\text{ball}(q, r)$ to denote the ball with center q and radius r .

In the following we show how to maintain a sketch $\Gamma(t)$ of $P(t)$ for the k -center problem with outliers. The idea behind our algorithm is as follows. Consider an optimal solution for $P(t)$ consisting of k balls B_1, \dots, B_k of radius ρ_{opt} . Suppose we cover the points in each ball B_i by a number of smaller balls of radius $\varepsilon \rho_{\text{opt}}$; we will call these *mini-balls*. Since the underlying space has doubling dimension d , we can do this using $O(1/\varepsilon^d)$ mini-balls for each ball B_i . To obtain a $(1 + \varepsilon)$ -approximation, we do not need to know all the points: it suffices to keep $z + 1$ points in each mini-ball, since then we cannot designate all points in the mini-ball as outliers. There are several challenges to overcome to make this idea work. For instance, we do not know ρ_{opt} . Hence, we will develop an algorithm for the decision version of the problem, which we will then apply “in parallel” to different possible values for the optimal radius. But when ρ is much smaller than ρ_{opt} we have another challenge, namely that there can be many outliers (which may need to be stored, because at some later moment in time they may become inliers). Finally, since points arrive and expire, the balls in an optimal solution move over time, which makes it hard to maintain the mini-balls correctly. Below we show how to overcome these challenges.

⁵ We allow the same point from X to occur multiple times in the stream, so $P(t)$ is actually a multi-set. Whenever we refer to “sets” in the remainder of the paper we mean “multi-sets”. The distance Δ_{\min} is defined with respect to distinct points, however.

2.1 A sketch for the decision problem

Recall that for a set Q of points, $\text{OPT}_{k,z}(Q)$ denotes the smallest radius ρ such that we can cover all points from Q , except for at most z outliers, by k balls of radius ρ , and that Δ_{\min} and Δ_{\max} are bounds on the minimum and maximum distance between any two points in the stream. Let ρ be a parameter with $\Delta_{\min} \leq \rho \leq \Delta_{\max}$. We will design a sketch $\Gamma(t)$ and a corresponding decision algorithm `TRYTOCOVER` with the following properties:

- The sketch $\Gamma(t)$ uses $O(kz/\varepsilon^d)$ storage.
- `TRYTOCOVER` either reports a collection \mathcal{C}^* of k balls of radius $\text{OPT}_{k,z}(P(t)) + 2\varepsilon\rho$ that together cover all points in $P(t)$ except for at most z outliers, or it reports `NO`. In latter case we have $\text{OPT}_{k,z}(P(t)) > 2\rho$.

Our sketch $\Gamma(t)$ is a tuple $[\tau(t), \mathcal{B}(t), \mathcal{R}(t), P_{\text{out}}(t)]$, where τ is a *timestamp*, $\mathcal{B}(t)$ is a collection of mini-balls, $\mathcal{R}(t)$ is a collection of so-called *representative sets*, and $P_{\text{out}}(t)$ is a set of outliers. We will maintain the following invariants.

- (Inv-1)** For any time t' with $t \leq t' < \tau(t)$ we are guaranteed that $\text{OPT}_{k,z}(P(t')) > 2\rho$. The idea is that when we find $k + z + 1$ points with pairwise distances greater 2ρ , then we can set τ to the expiration time of the oldest of these points, and we can delete this point (as well as any older points) from the sketch.
- (Inv-2)** Each mini-ball $B \in \mathcal{B}(t)$ has radius $\varepsilon\rho$ and the center of B , denoted by $\text{center}(B)$, is a point from the stream. Note that the center does not need to be a point from $P(t)$. The mini-balls in $\mathcal{B}(t)$ are *well spread* in the sense that no mini-ball contains the center of any other mini-ball. In other words, we have $\text{dist}(\text{center}(B), \text{center}(B')) > \varepsilon\rho$ for any two mini-balls $B, B' \in \mathcal{B}(t)$.
- (Inv-3)** For each mini-ball $B \in \mathcal{B}(t)$ the set $\mathcal{R}(t)$ contains a *representative set* $R(B) \subseteq B \cap P(t)$, and these are the only sets in $\mathcal{R}(t)$. The representative sets $R(B)$ are pairwise disjoint, and each set $R(B)$ contains at most $z + 1$ points. When $|R(B)| = z + 1$, we say that the mini-ball B is *full*.
- (Inv-4)** Define $P_{\text{in}}(t) := \bigcup_{B \in \mathcal{B}(t)} R(B)$ and let $S(t) := P_{\text{in}}(t) \cup P_{\text{out}}(t)$ be the collection of all points in our sketch. For any point $p_i \in P(t) \setminus S(t)$ – these are exactly the points that have not yet expired but that have been discarded by the algorithm – we have (i) $t_{\text{exp}}(p_i) \leq \tau(t)$, and/or (ii) $p_i \in B$ for a mini-ball $B \in \mathcal{B}(t)$ that is full and such that all points in $R(B)$ arrived after p_i .
- (Inv-5)** $|\mathcal{B}(t)| = O(k/\varepsilon^d)$ and $|P_{\text{out}}(t)| \leq z$.

At time $t = 0$, before the arrival of the first point, we have $\tau(t) = 0$ and $\mathcal{B}(t) = \mathcal{R}(t) = P_{\text{out}}(t) = \emptyset$. Since $P(0) = \emptyset$ this trivially satisfies the invariants. Before we prove that we can maintain the sketch upon the arrival and departure of points, we present our decision algorithm `TRYTOCOVER` and prove its correctness.

Algorithm 1 TRYTOCOVER($\Gamma(t)$).

- 1: $S(t) \leftarrow \bigcup_{B \in \mathcal{B}(t)} R(B)$
 - 2: Compute $\text{OPT}_{k,z}(S(t))$ and the corresponding collection $\mathcal{C} := \{C_1, \dots, C_k\}$ of balls.
 - 3: **if** $t < \tau(t)$ or $\text{OPT}_{k,z}(S(t)) > 2\rho$ **then**
 - 4: Report NO
 - 5: **else**
 - 6: Increase the radius of each ball $C_i \in \mathcal{C}$ by $2\varepsilon\rho$.
 - 7: Report the collection $\mathcal{C}^* := \{C_1^*, \dots, C_k^*\}$ of expanded balls.
-

In line 2 we compute an optimal solution on the point set $S(t)$. How this can be done depends on the underlying metric space and will be discussed in Section 2.3. The following lemma establishes the correctness of the algorithm.

► **Lemma 1.** *Algorithm TRYTOCOVER either reports a collection \mathcal{C}^* of k balls of radius $\text{OPT}_{k,z}(P(t)) + 2\varepsilon\rho$ that together cover all points in $P(t)$ except for at most z outliers, or it reports NO. In latter case we have $\text{OPT}_{k,z}(P(t)) > 2\rho$.*

Proof. First suppose the algorithm reports NO. If this happens because $t < \tau(t)$ then $\text{OPT}_{k,z}(P(t)) > 2\rho$ by (Inv-1). Otherwise, this happens because $\text{OPT}_{k,z}(S(t)) > 2\rho$. But then $\text{OPT}_{k,z}(P(t)) > 2\rho$, because (Inv-3) implies that $S(t) \subseteq P(t)$.

Now suppose the algorithm reports a collection $\mathcal{C}^* := \{C_1^*, \dots, C_k^*\}$ of balls. Let \mathcal{C} be the corresponding set of balls before they were expanded. Since \mathcal{C} is an optimal solution for $S(t)$, the balls C_i have radius $\text{OPT}_{k,z}(S(t)) \leq \text{OPT}_{k,z}(P(t))$ and together they cover all points in $S(t)$ except for at most z outliers. Now consider a point $p_i \in P(t) \setminus S(t)$. To finish the proof, we must show that p_i is covered by one of the balls in \mathcal{C}^* . To this end, first observe that $t_{\text{exp}}(p_i) > t$ because $p_i \in P(t)$. Since TRYTOCOVER did not report NO, this implies that $t_{\text{exp}}(p_i) > \tau(t)$. Hence, we can conclude from (Inv-4) that $p_i \in B$ for a mini-ball $B \in \mathcal{B}(t)$ that is full. Thus $R(B)$ contains $z + 1$ points, and since we allow only z outliers this implies that at least one point from $R(B)$ is covered by a ball $C_i \in \mathcal{C}$. Because $\text{diam}(B) = 2\varepsilon\rho$, this implies that p_i must be covered by C_i^* , thus finishing the proof. ◀

Next we show how to update the sketch $\Gamma(t)$.

Handling departures

When a point p_j in one of our representative sets $R(B)$ expires, we simply delete it from $R(B)$, and if $R(B)$ then becomes empty we remove $R(B)$ from $\mathcal{R}(t)$ and B from $\mathcal{B}(t)$. Similarly, if p_j was a point in $P_{\text{out}}(t)$ we remove p_j from $P_{\text{out}}(t)$.

It is trivial to verify that (Inv-1)–(Inv-3) and (Inv-5) still hold for the updated sketch. To see that (Inv-4) holds as well, consider a point $p_i \in P(t) \setminus S(t)$. The only reason for (Inv-4) to be violated, would be when $p_i \in B$ for a mini-ball B that was full before the deletion of p_j but is no longer full after the deletion. However, (Inv-4) states that all points in $R(B)$ arrived after p_i . Since p_i did not yet expire, this means that the point p_j that currently expires cannot be a point from $R(B)$.

Handling arrivals

The arrival of a point p_j at time $t := t_{\text{arr}}(p_j)$ is handled by Algorithm 2. We denote the sketch just before the arrival by $\Gamma(t^-)$, and the updated sketch by $\Gamma(t^+)$.

Algorithm 2 HANDLEARRIVAL($\Gamma(t^-), p_j$).

1. If p_j lies in an existing mini-ball $B \in \mathcal{B}(t^-)$ then add p_j to $R(B)$. If p_j did not lie in an existing mini-ball then add p_j to $P_{\text{out}}(t^-)$.
 2. Let $Q := P_{\text{out}}(t^-) \cup \bigcup_{B \in \mathcal{B}(t^-)} R(B)$ be the set of all points in the current sketch, including the just inserted point. Sort the points in Q by decreasing arrival time, and let $q_1, q_2 \dots$ denote the sorted sequence of points – note that q_1 is the newly inserted point p_j . Let $Q_i := \{q_1, \dots, q_i\}$ denote the set containing the first i points from Q . Find the largest index i^* such that the following holds:
 - The 3-approximation algorithm of Charikar et al. [7] for the k -center clustering algorithm with z outliers, when run on Q_{i^*} , reports a collection \mathcal{B}^* of k balls of radius at most 6ρ . (In fact, any other constant-factor approximation algorithm can be used if we adjust the value 6ρ appropriately. For concreteness, we use the one by Charikar et al.)
 3. If $Q_{i^*} \neq Q$ (and so the point q_{i^*+1} exists) then set $\tau(t^+) := \max(\tau(t^-), t_{\text{exp}}(q_{i^*+1}))$. Remove the points in $Q \setminus Q_{i^*}$ from the representative sets they are in; note that these points all have expiration time at least $\tau(t^+)$.
 4. Let $\mathcal{B}^* := \{B_1, \dots, B_k\}$ be the balls in the solution reported for Q_{i^*} . Note that all points from Q_{i^*} , except for at most z outliers, lie in a ball from \mathcal{B}^* . The new set of mini-balls and outliers is now computed as follows. Increase the radius of the balls in \mathcal{B}^* by $\varepsilon\rho$. Denote the set of expanded balls by $\overline{\mathcal{B}}^*$
 - a. Add each mini-ball $B \in \mathcal{B}(t^-)$ whose center lies inside a ball from $\overline{\mathcal{B}}^*$ to the set $\mathcal{B}(t^+)$ and add their representative sets to $\mathcal{R}(t^+)$.
 - b. Let $Z \subset Q_{i^*}$ be the set of points that are not in a representative set added to $\mathcal{R}(t^+)$ in Step 4a. Let $Z_{\text{in}} \subseteq Z$ be the points that lie inside a ball from $\overline{\mathcal{B}}^*$. Go over the points in Z_{in} one by one, in arbitrary order, and handle each point q as follows: If there is a mini-ball in the current set $\mathcal{B}(t^+)$ that contains q then add q to $R(B)$ for an arbitrary such mini-ball B ; otherwise create a mini-ball B with center q and radius $\varepsilon\rho$, set $R(B) := \{q\}$ and add B to $\mathcal{B}(t^+)$.
 - c. For all mini-balls $B \in \mathcal{B}(t^+)$ such that $|R(B)| > z+1$, remove the oldest $|R(B)| - (z+1)$ points from $R(B)$.
 - d. Set $P_{\text{out}}(t^+) := Z \setminus Z_{\text{in}}$. Note that these are exactly the points from Q_{i^*} that have not been added to a representative set
-

► **Lemma 2.** *After the arrival of point p_j has been handled, the invariants (Inv-1)–(Inv-5) are restored.*

Proof. (Inv-1) If $\tau(t^+) = \tau(t^-)$ then obviously (Inv-1) still holds, so assume that τ is updated by the algorithm in Step 3. Thus the algorithm of Charikar et al. [7] returned a solution of radius more than 6ρ on Q_{i^*+1} . Since the algorithm is a 3-approximation, this means $\text{OPT}_{k,z}(Q_{i^*+1}) > 2\rho$. Because we sorted the points on expiration time, we must have $\text{OPT}_{k,z}(P(t')) > 2\rho$ for all t' with $t \leq t' < t_{\text{exp}}(q_{i^*+1})$. Hence, (Inv-1) still holds.

(Inv-2) By construction, the mini-balls in $\mathcal{B}(t^+)$ have radius $\varepsilon\rho$ and are centered at a point from the stream. The mini-balls from $\mathcal{B}(t^-)$ are well spread by induction, so we only need to prove that any mini-balls created in Step 4b are well spread. This is true because we only create a new mini-ball when its center does not lie inside an existing mini-ball.

(Inv-3) The representative sets $R(B)$ are disjoint by construction, and $|R(B)| \leq z+1$ is guaranteed by Step 4c. Hence (Inv-3) holds.

(Inv-4) Take a point $p_i \in P(t^+) \setminus S(t^+)$ and assume $t_{\text{exp}}(p_i) > \tau(t^+)$. We have two cases.

- The first case is that p_i was discarded due to the arrival of p_j . Then it was either removed in Step 3, which implies $t_{\text{exp}}(p_i) \geq \tau(t^+)$; or it was removed in Step 4c, which implies there is a full mini-ball $B \in \mathcal{B}(t^+)$ such that all points in $R(B)$ are newer than p_i . Hence, (Inv-4) holds for p_i .
- The second case is that p_i was already discarded earlier. We can assume that (Inv-4) holds before the arrival of p_j . If $t_{\text{exp}}(p_i) \leq \tau(t^-)$ then we also have $t_{\text{exp}}(p_i) \leq \tau(t^+)$ and we are done. So assume that $p_i \in B$ for a mini-ball $B \in \mathcal{B}(t^-)$ that was full and such that all points in $R(B)$ arrived after p_i . If $R(B)$ contained a point $p_{i'} \in Q \setminus Q_{i^*}$ then $t_{\text{exp}}(p_i) < t_{\text{exp}}(p_{i'}) \leq \tau(t^+)$, and so (Inv-4) holds. If $R(B)$ only contained points from Q_{i^*} then we have two sub-cases.

The first sub-case is that B is still present in $\mathcal{B}(t^+)$. Then B must still be full, because $R(B)$ only contained points from Q_{i^*} . Hence, (Inv-4) holds.

The second sub-case is that B is not present in $\mathcal{B}(t^+)$. Thus its center must lie outside all balls in $\bar{\mathcal{B}}^*$. But then the points in $R(B)$, which are all in Q_{i^*} , are outside any ball in \mathcal{B}^* . But this is impossible, since $|R(B)| = z + 1$ and \mathcal{B}^* has only z outliers.

(Inv-5) All mini-balls in $\mathcal{B}(t^+)$ have their center inside one of the k balls in $\bar{\mathcal{B}}^*$, which have radius at most 6ρ . Moreover, the mini-balls in $\mathcal{B}(t^+)$ have radius $\varepsilon\rho$ and they are well spread. Since the underlying space has doubling dimension d , this implies that $|\mathcal{B}(t^+)| = O(k/\varepsilon^d)$.

The outlier set $P_{\text{out}}(t^-)$ consists of the points from Q_{i^*} that lie outside the balls in $\bar{\mathcal{B}}^*$. Hence, these points also lie outside the balls in \mathcal{B}^* . Since \mathcal{B}^* is a valid solution to the k -center problem with z outliers, there are at most z such points. Thus $|P_{\text{out}}(t^+)| \leq z$. ◀

2.2 A sketch for the optimization problem

Above we presented a sketch for a decision version of the problem, for given parameters ρ and ε . The sketch uses $O(kz/\varepsilon^d)$ storage. We also gave an algorithm `TRYTOCOVER` that either reports a collection \mathcal{C}^* of k balls of radius $\text{OPT}_{k,z}(P(t)) + 2\varepsilon\rho$ covering all points in $P(t)$ except at most z outliers, or that reports NO. In latter case we know that $\text{OPT}_{k,z}(P(t)) > 2\rho$. To make the parameter ρ and ε explicit, we will from now on denote the sketch by $\Gamma_{\rho,\varepsilon}$.

In the optimization version of the problem we wish to find k congruent balls of minimum radius that together cover all points in $P(t)$ except for at most z outliers. To obtain a $(1 + \varepsilon)$ -approximation for the optimization problem, for a given $\varepsilon > 0$, we maintain a sketch $\Gamma_{\rho_i,\varepsilon/2}$ for every $0 \leq i \leq \lfloor \log \sigma \rfloor$, where $\rho_i := 2^i \cdot \Delta_{\min}$. We can then obtain a $(1 + \varepsilon)$ -approximate solution with the algorithm shown in Algorithm 3.

■ **Algorithm 3** FINDAPPROXIMATECENTERS($\Gamma(t)$).

```

1:  $i \leftarrow 0$ 
2: repeat
3:    $\text{answer} \leftarrow \text{TRYTOCOVER}(\Gamma_{\rho_i,\varepsilon/2})$ ;  $i \leftarrow i + 1$ 
4: until  $\text{answer} \neq \text{NO}$ 
5: if the radii of the balls in  $\text{answer}$  is less than  $\Delta_{\min}$  then
6:   Reduce the radii of the balls to zero
7: Report  $\text{answer}$ 

```

We obtain the following theorem, whose proof can be found in the full version of the paper.

► **Theorem 3.** *The sketch above uses $O((kz/\varepsilon^d) \log \sigma)$ storage and that allows us to report at any time t a collection \mathcal{C}^* of balls of radius at most $(1 + \varepsilon) \cdot \text{OPT}_{k,z}(P(t))$ covering all points from $P(t)$, except at most z outliers.*

Our sketch for the k -center problem can also be used for the k' -center problem for $k' < k$. Moreover, a sketch for the k -center problem for $k \geq 1$ can be used for the diameter problem. Recall that the diameter problem with outliers for the set $P(t)$ asks for the value

$$\text{diam}_z(P(t)) := \min\{\text{diam}(P(t) \setminus Q) : |Q| = z\},$$

that is, $\text{diam}_z(P(t))$ is the smallest diameter one can obtain by deleting z outliers from $P(t)$. We say that an algorithm reports a $(1 + \varepsilon)$ -approximation to $\text{diam}_z(P(t))$ if it reports a value D with $(1 + \varepsilon) \cdot \text{diam}_z(P(t)) \leq D \leq \text{diam}_z(P(t))$. The proof of the following theorem is in the full version of the paper.

► **Theorem 4.** *The sketch for the k -center problem with outliers as presented above can also be used to provide a $(1 + \varepsilon)$ -approximation for the k' -center problem with outliers, for any $1 \leq k' \leq k$. Moreover, it can be used to provide a $(1 + 2\varepsilon)$ -approximation for the diameter problem with outliers.*

Theorem 4 implies that there exists a sketch for the diameter problem with outliers in the sliding-window model that gives a $(1 + \varepsilon)$ -approximation to $\text{diam}_z(P(t))$ using $O((z/\varepsilon^d) \log \sigma)$ storage, namely the sketch for the 1-center problem.

2.3 Time complexity

Above we focused on the storage used by our sketch, and on the approximation ratio it can potentially provide. We now discuss the time complexity.

The algorithm that handles departures trivially runs in $O(1)$ time. The most time-consuming step in the algorithm that handles arrivals is Step 2, where we run the algorithm of Charikar et al. [7] on a set of $O(kz/\varepsilon^d)$ points. This takes $O((kz/\varepsilon^d)^3)$ time. Note that, both for departures and arrivals, we need to execute the respective algorithms for each of the $\log \sigma$ sketches $\Gamma_{\rho_i, \varepsilon/2}$ we maintain.

The time needed for FINDAPPROXIMATECENTERS is more interesting, since it calls TRYTOCOVER, which is supposed to compute an optimal solution for the k -center problem with outliers on a given set $S(t)$ of $O(kz/\varepsilon)$ points. It is routine to check that if TRYTOCOVER would use a c -approximate solution on $S(t)$, instead of an optimal solution, our final result would be a $c(1 + \varepsilon)$ -approximation. In d -dimensional Euclidean space we can solve the k -center problem with outliers in time polynomial in $n := |S(t)|$ (for constant k and d), as follows: first generate all $O(n^{d+1})$ potential centers – there are $O(n^{d+1})$ potential centers because the smallest enclosing ball of a given point set is defined by at most $d + 1$ points – then generate all possible collections of k such centers, and then for each of the $O(n^{(d+1)k})$ such collections find the minimum radius ρ such that we can cover all except for z points. A similar approach is possible for other metrics in \mathbb{R}^d that are sufficiently well-behaved (such as ℓ_p metrics, for instance).

For arbitrary metric spaces the situation is more tricky. The standard assumption, which we also make, is that we can compute the distance between any two given points from the metric space in $O(1)$ time. Still, computing an optimal solution on the set $S(t)$ can be quite slow or perhaps even infeasible. For example, when an optimal center is a point from the metric space that does not occur in the stream, then the algorithm may be unable to retrieve this center. So unless we are in Euclidean space, or some other “well-defined” space, it

seems most natural to define the underlying space as consisting of exactly the points in the stream. Note that this can still be somewhat problematic, as it may require the algorithm to use points that have already expired as centers for the current set $S(t)$. These issues seem unavoidable for any algorithm that maintains a sketch in the sliding-window model. Also note that, using just the points in the current set $P(t)$ as centers, one can always obtain a 2-approximation to the optimal clustering. Invariant (Inv-4) thus implies that by using the set $S(t)$ as potential centers we can obtain a $(2 + \varepsilon)$ approximation. Also note that the optimal k -center clustering with z outliers on $S(t)$, under the restriction that the centers come from $S(t)$, can be trivially computed in $O(|S(t)|^{k+2})$ time.

The following theorem summarizes the performance of our sketch.

► **Theorem 5.** *Consider the k -center problem with z outliers in the sliding-window model, for streams of points from a space with doubling dimension d . Let Δ_{\min} and Δ_{\max} be given (lower and upper) bounds on the minimum and maximum distance between any two points in the stream, and let $\sigma := \Delta_{\max}/\Delta_{\min}$ be (an upper bound on) the spread of the stream. Suppose we have a c -approximation algorithm for the static version of the problem that runs in $T_{k,z}(n)$ time on a set of n points. Then for any $0 < \varepsilon < 1$ we can maintain a sketch with the following properties:*

- *The sketch uses $O((kz/\varepsilon^d) \log \sigma)$ storage.*
- *Departures and arrivals can be handled in $O(\log \sigma)$ and $O((kz/\varepsilon^d)^3 \log \sigma)$ time, respectively.*
- *At any time t , the algorithm can report a valid solution for $P(t)$ of cost at most $(1 + \varepsilon)c \cdot \text{OPT}_{k,z}(P(t))$. The time needed to compute the solution is $T_{k,z}(n)$, where $n := O(kz/\varepsilon^d)$.*

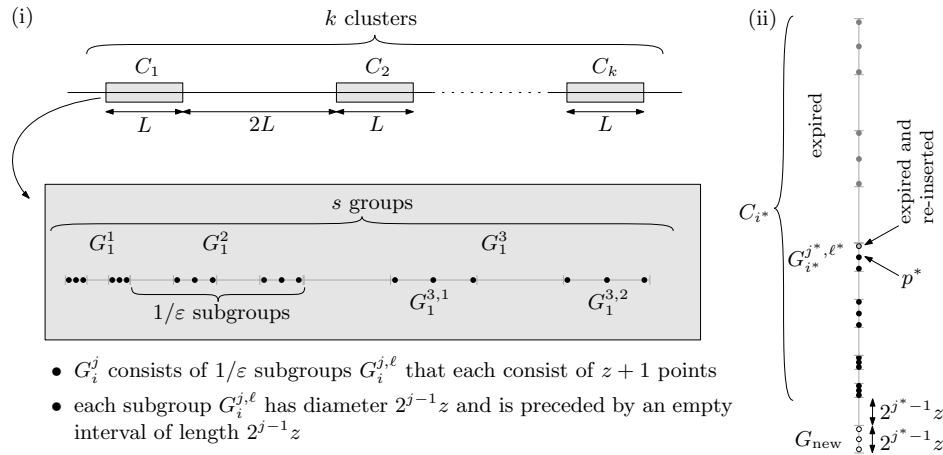
3 A lower bound

Above we presented a sketch of size $O((kz/\varepsilon^d) \log \sigma)$ that provides a $(1 + \varepsilon)$ -approximation for the k -center problem with z outliers in the sliding-window model. In this section we show that this is tight, up to the dependency on d . In particular, we prove that any sketch giving a $(1 + \varepsilon)$ -approximation for the problem in \mathbb{R}^1 must use $\Omega((kz/\varepsilon) \log \sigma)$ storage. In the full paper we show that to obtain any constant approximation ratio, a sketch must use $\Omega(kz)$ storage.

The lower-bound model

Our lower-bound is extremely simple and general. We allow the algorithm to store points, or weighted points, or balls, or whatever it wants so that it can approximate an optimal solution for $P(t)$ at any time t . We only make the following assumptions. Let $S(t)$ be the collection of objects being stored at time t .

- Each object in $S(t)$ is accompanied by an expiration time, which is equal to the expiration time of some point $p_i \in P(t)$.
- Let $p_i \in P(t)$. If no object in $S(t)$ uses $t_{\text{exp}}(p_i)$ as its expiration time, then no object in $S(t')$ with $t' > t$ can use $t_{\text{exp}}(p_i)$ as its expiration time.
- The solution reported by the algorithm is uniquely determined by $S(t)$, and the algorithm only modifies $S(t)$ when a new point arrives or when an object in $S(t)$ expires.
- The algorithm is deterministic and oblivious of future arrivals. In other words, the set $S(t)$ is uniquely determined by the sequence of arrivals up to time t , and the solution reported for $P(t)$ is uniquely determined by $S(t)$.



■ **Figure 1** The lower-bound construction for a $(1 + \varepsilon)$ -approximate sketch. In the example, $s = 3$, $k = 3$, $\varepsilon = 1/2$ and $z = 2$. (i) The initial configuration. (ii) Rotated view of the cluster containing p^* .

The storage used by the algorithm is defined as the number of objects in $S(t)$. The algorithm can decide which objects to keep in $S(t)$ anyway it wants; it may even keep an unbounded amount of extra information in order to make its decisions. The algorithm can also derive a solution for $P(t)$ in any way it wants, as long as the solution is valid and uniquely determined by $S(t)$. Clearly, the sketch from the previous section adheres to the model.

A lower bound for a $(1 + \varepsilon)$ -approximate sketch

Let ALG be a $(1 + \varepsilon')$ -approximation algorithm, for some $0 < \varepsilon' < 1$, where we assume for simplicity that $1/\varepsilon'$ is an integer. The lower-bound instance, which consists of points in \mathbb{R}^1 , is as follows.

Let $\varepsilon := \varepsilon'/8$ and consider the configuration shown in Figure 1(i). The configuration consists of k clusters C_1, \dots, C_k , placed next to each other from left to right. Each cluster C_i consists of s groups, G_1^i, \dots, G_s^i , where $s = \Theta(\log \sigma)$. Each group G_j^i consists of $1/\varepsilon$ subgroups, $G_1^{j,i}, \dots, G_{1/\varepsilon}^{j,i}$. A subgroup $G_{i^{\ell}}^{j,\ell}$ consists of $z+1$ points at distance 2^{j-1} apart; the diameter of a subgroup – that is, the distance between its leftmost and rightmost point – is thus $2^{j-1}z$. Subgroup $G_{i^{\ell}}^{j,\ell}$ is preceded by an empty interval of length $2^{j-1}z$. Hence, the total diameter of the group G_i , including the empty interval preceding its leftmost subgroup, is $2^{i-1}z/\varepsilon$. (The exception is G_1 , for which we do not count the empty interval preceding its leftmost subgroup.) This brings the total diameter of a cluster to

$$L := \sum_{j=1}^s 2^j z/\varepsilon - z/\varepsilon = (2^{s+1} - 3) \cdot z/\varepsilon < 2^{s+1} z/\varepsilon.$$

In between every two consecutive clusters there is an empty interval of length $2L$.

The arrival order of the points in the configuration is as follows. The subgroups within a cluster arrive from right to left, in a round-robin fashion over the clusters: in the first round the subgroups $G_k^{s, \frac{1}{\varepsilon}}, \dots, G_1^{s, \frac{1}{\varepsilon}}$ arrive, in the second round the subgroups $G_k^{s, \frac{1}{\varepsilon}-1}, \dots, G_1^{s, \frac{1}{\varepsilon}-1}$ arrive, and so on. More formally, $G_{i^{\ell}}^{j,\ell}$ arrives before $G_{i'^{\ell'}}^{j',\ell'}$ iff: $j > j'$ or $(j = j' \text{ and } \ell > \ell')$ or $(j = j' \text{ and } \ell = \ell' \text{ and } i > i')$. This finishes the description of the initial part of the instance.

Let t be the time at which the last point in the configuration arrives, let $P(t)$ be the set of all points in the clusters C_1, \dots, C_k , and let $S(t)$ be the set of objects stored by ALG. We will argue that $|S(t)| = \Omega(kzs/\varepsilon)$, otherwise an adversary can continue the instance such that at some time t' in the future the algorithm does not give a $(1 + \varepsilon)$ -approximation.

Consider the points in the subgroups $G_i^{j,\ell}$ with $j > 1$, except for the very first point that arrived. Let T_{exp} be the set of expiration times of these points. Note that $|T_{\text{exp}}| = (k(z+1)(s-1)/\varepsilon) - 1$. Suppose at least one of these expiration times, say the expiration time of point p^* , is not used by any object in $S(t)$. Let $t_{p^*}^-$ and $t_{p^*}^+$ be the times immediately before and after $t_{\text{exp}}(p^*)$, respectively. The conditions of the model imply that the algorithm reports the same solution at times $t_{p^*}^-$ and $t_{p^*}^+$. The adversary will now continue the scenario in such that one of these answers is not sufficiently accurate, as described next.

Let $G_{i^*}^{j^*,\ell^*}$ be the subgroup that p^* belongs to. First, the adversary waits until all points that arrived before p^* have expired. The situation is then as follows: clusters C_i with $i < i^*$ consist of the subgroups up to $G_{j^*,\ell^*}^{(i)}$, while clusters C_i with $i > i^*$ are missing the last of these subgroups. Cluster C_{i^*} has the points up to point p^* (which expires next) in $G_{i^*}^{j^*,\ell^*}$. Before p^* expires, the adversary now adds a new subgroup G_{new} of $z+1$ points to C_{i^*} . The diameter of G_{new} is $z^{j^*-1}z$ and its distance to C_{i^*} is $z^{j^*-1}z$ as well; see Figure 1(ii). (The construction has been rotated by 90 degrees in the figure, to make it fit.) In addition, the adversary adds the points from $G_{j^*,\ell^*}^{(i)}$ that have already expired back.

We now analyze $\text{OPT}_{k,z}(P(t_{p^*}^-))$. Since the distance between any two of the k clusters is larger than the diameter of the clusters, and any cluster still contains at least $z+1$ points – the latter follows because $j^* > 1$ – an optimal solution will use a separate ball for each cluster. The largest cluster is C_{i^*} , because G_{new} was added to it. Since each subgroup in C_{i^*} has $z+1$ points and we can designate only z points as outliers, it is optimal to designate the z rightmost points (topmost in Figure 1(ii)) from $G_{i^*}^{j^*,\ell^*}$ as outliers and cover the remaining points with a ball of diameter $L^* + 3 \cdot 2^{j^*-1}z$, where

$$L^* := \sum_{j=1}^{j^*-1} 2^j z/\varepsilon - z/\varepsilon + 2\ell^* \cdot 2^{j^*-1}z = (2^{j^*} - 3) z/\varepsilon + \ell^* 2^{j^*} z < (2^{j^*+1}) z/\varepsilon,$$

where the last inequality uses that $\ell^* \leq 1/\varepsilon$. Hence, $\text{OPT}_{k,z}(P(t_{p^*}^-)) = (L^* + 3 \cdot 2^{j^*-1}z)/2$.

At time $t_{p^*}^+$ the point p^* expires. Hence, at time $t_{p^*}^+$ the subgroup $G_{i^*}^{j^*,\ell^*}$ has only z points, which can all be designated as outliers. Hence, C_{i^*} minus the outliers can be covered by a ball of diameter $L^* + 2 \cdot 2^{j^*-1}z$ and so $\text{OPT}_{k,z}(P(t_{p^*}^+)) = (L^* + 2 \cdot 2^{j^*-1}z)/2$.

Since the algorithm must report a valid solution at time $t_{p^*}^-$ and it must report the same solution at time $t_{p^*}^+$, the approximation ratio at time $t_{p^*}^+$ is at least

$$\frac{\text{OPT}_{k,z}(P(t_{p^*}^-))}{\text{OPT}_{k,z}(P(t_{p^*}^+))} \geq \frac{L^* + 3 \cdot 2^{j^*-1}z}{L^* + 2 \cdot 2^{j^*-1}z} = 1 + \frac{2^{j^*-1}z}{L^* + 2^{j^*}z} > 1 + \frac{2^{j^*-1}z}{2^{j^*+2}z/\varepsilon} = 1 + \varepsilon/8$$

We conclude that any algorithm that gives a $(1 + \varepsilon')$ -approximation must store at least $k(z+1)(s-1)/(8\varepsilon')$ objects in the worst case.

The spread of the point set used in the construction, including the subgroup G_{new} , is⁶

$$\sigma = kL + (k-1)2L + 2^{j^*}z < 3kL < 3k \cdot 2^{s+1}z/\varepsilon.$$

⁶ In our definition of spread, we are allowed to reuse points, since $\Delta_{\min} = 0$ is defined as the minimum distance between distinct points in the stream. We can also avoid reusing points in our lower-bound scenario, without asymptotically influencing the spread of the points.

If the spread is not too small, namely when $\sigma \geq (3kz/\varepsilon)^2$, we have $s \geq \frac{1}{2} \log \sigma - 1$. We obtain the following theorem.

► **Theorem 6.** *Let ALG be a $(1 + \varepsilon)$ -approximation algorithm for the k -center problem with z outliers in \mathbb{R}^1 , with $z \geq 1$, that works in the model described above. For any spread $\sigma \geq (3kz/\varepsilon)^2$, there is a problem instance of spread σ that requires ALG to store $\Omega((kz \log \sigma)/\varepsilon)$ points.*

References

- 1 Pankaj K. Agarwal, Sarel Har-Peled, and Hai Yu. Robust shape fitting via peeling and grating coresets. *Discret. Comput. Geom.*, 39(1-3):38–58, 2008. doi:10.1007/s00454-007-9013-2.
- 2 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015. doi:10.1007/s00453-013-9846-4.
- 3 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k -center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially. *Proc. VLDB Endow.*, 12(7):766–778, 2019. doi:10.14778/3317315.3317319.
- 4 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom.*, 47(2):240–247, 2014. doi:10.1016/j.comgeo.2013.05.007.
- 5 Timothy M. Chan and Bashir S. Sadjad. Geometric optimization problems over sliding windows. *Int. J. Comput. Geom. Appl.*, 16(2-3):145–158, 2006. doi:10.1142/S0218195906001975.
- 6 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi:10.1137/S0097539702418498.
- 7 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proc. 12th Annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.
- 8 Moses Charikar, Liadan O’Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003. doi:10.1145/780542.780548.
- 9 Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k -center in sliding windows. In *Proc. 43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 55 of *LIPIcs*, pages 19:1–19:12, 2016. doi:10.4230/LIPIcs.ICALP.2016.19.
- 10 Hu Ding, Haikuo Yu, and Zixiu Wang. Greedy strategy works for k -center clustering with outliers and coreset construction. In *Proc. 27th Annual European Symposium on Algorithms (ESA)*, volume 144 of *LIPIcs*, pages 40:1–40:16, 2019. doi:10.4230/LIPIcs.ESA.2019.40.
- 11 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005. doi:10.1007/s00453-004-1105-2.
- 12 Behnam Hatami and Hamid Zarrabi-Zadeh. A streaming algorithm for 2-center with outliers in high dimensions. *Comput. Geom.*, 60:26–36, 2017. doi:10.1016/j.comgeo.2016.07.002.
- 13 Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k -center clustering with outliers and with anonymity. In *Proc. 11th and 12th International Workshop on Approximation, Randomization and Combinatorial Optimization (APPROX and RANDOM)*, volume 5171 of *Lecture Notes in Computer Science*, pages 165–178, 2008. doi:10.1007/978-3-540-85363-3_14.
- 14 Hamid Zarrabi-Zadeh. Core-preserving algorithms. In *Proceedings of the 20th Annual Canadian Conference on Computational Geometry (CCCG)*, 2008.
- 15 Hamid Zarrabi-Zadeh and Asish Mukhopadhyay. Streaming 1-center with outliers in high dimensions. In *Proc. 21st Annual Canadian Conference on Computational Geometry (CCCG)*, pages 83–86, 2009. URL: http://cccg.ca/proceedings/2009/cccg09_22.pdf.

Incremental SCC Maintenance in Sparse Graphs

Aaron Bernstein ✉

Rutgers University, New Brunswick, NJ, USA

Aditi Dudeja ✉

Rutgers University, New Brunswick, NJ, USA

Seth Pettie ✉

University of Michigan, Ann Arbor, MI, USA

Abstract

In the *incremental cycle detection* problem, edges are added to a directed graph (initially empty), and the algorithm has to report the presence of the first cycle, once it is formed. A closely related problem is the *incremental topological sort* problem, where edges are added to an acyclic graph, and the algorithm is required to maintain a valid topological ordering. Since these problems arise naturally in many applications such as scheduling tasks, pointer analysis, and circuit evaluation, they have been studied extensively in the last three decades. Motivated by the fact that in many of these applications, the presence of a cycle is not fatal, we study a generalization of these problems, *incremental maintenance of strongly connected components* (incremental SCC).

Several incremental algorithms in the literature which do cycle detection and topological sort in directed acyclic graphs, such as those by [7] and [16], also generalize to maintain strongly connected components and their topological sort in general directed graphs. The algorithms of [16] and [7] have a total update time of $O(m^{3/2})$ and $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$ respectively, and this is the state of the art for incremental SCC. But the most recent algorithms for incremental cycle detection and topological sort ([8] and [10]), which yield total (randomized) update time $\tilde{O}(\min\{m^{4/3}, n^2\})$, do not extend to incremental SCC. Thus, there is a gap between the best known algorithms for these two closely related problems.

In this paper, we bridge this gap by extending the framework of [10] to general directed graphs. More concretely, we give a Las Vegas algorithm for incremental SCCs with an expected total update time of $\tilde{O}(m^{4/3})$. A key ingredient in the algorithm of [10] is a structural theorem (first introduced in [8]) that bounds the number of “equivalent” vertices. Unfortunately, this theorem only applies to DAGs. We show a natural way to extend this structural theorem to general directed graphs, and along the way we develop a significantly simpler and more intuitive proof of this theorem.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Directed Graphs, Strongly Connected Components, Dynamic Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.14

Funding Aaron Bernstein: This work was done while funded by NSF Award 1942010.

1 Introduction

In dynamic algorithms, our main goal is to maintain a key property of the graph while an adversary makes changes in the graph in the form of edge insertions and deletions. An algorithm is called incremental if it handles only insertions, decremental if it handles only deletions and fully dynamic if it handles both insertions as well as deletions. For a dynamic algorithm we hope to optimize the update time of the algorithm, which is the time taken by the algorithm to adapt to the changes to the input and modify the results. For incremental/decremental algorithms, one typically seeks to minimize the *total* update time over the entire sequence of edge insertions/deletions.



© Aaron Bernstein, Aditi Dudeja, and Seth Pettie;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, 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

In this paper, we consider the problem of maintaining strongly connected components in the incremental setting (incremental SCC). This is a generalization of the problems of incremental cycle detection and topological sorting in directed acyclic graph, which find application in pointer analysis, deadlock detection [4], circuit evaluation [3] and scheduling tasks. In many of these applications, the presence of a cycle is not fatal, which motivates the general problem of maintaining strongly connected components, as well as the topological order of these components.

The problems of incremental cycle detection and topological sorting were first studied by Katriel and Bodlaender [18], who gave the first non-trivial algorithm for these problems with a total update time of $O(\min\{m^{3/2} \log n, m^{3/2} + n^2 \log n\})$. This bound was improved by Liu and Chao [20] to $O(m^{3/2} + m\sqrt{n} \log n)$. Since then, these problems and incremental SCC have been studied extensively (see for example [1, 2, 6, 16, 7, 13, 21]). Several algorithms that do incremental cycle detection and topological sort maintenance in directed acyclic graphs can be modified to get algorithms for incremental SCC. For example, the algorithm of Haeupler, Kavitha, Mathew, Sen and Tarjan [16] is able to do cycle detection as well as strongly connected component maintenance in $O(m^{3/2})$ total update time. In an important result, Bender, Fineman, Gilbert and Tarjan presented two algorithms for strongly connected components, with total update times of $O(n^2 \log n)$ and $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$, for dense and sparse graphs, respectively (see Table 1).

The two most recent algorithms in this area are limited to cycle detection and topological sort: Bernstein and Chechik [8] gave a Las Vegas algorithm with an expected total update time of $O(m\sqrt{n} \log n)$; Bhattacharya and Kulkarni [10] combined the balanced search approach of [16] with the results of [8] to get an algorithm with a total expected runtime of $\tilde{O}(m^{4/3})$. As a result, there was still a gap between the best known algorithms for cycle detection and topological sort (update time of $\tilde{O}(\min\{m^{4/3}, n^2\})$) and for incremental SCC (update time of $\tilde{O}(\min\{m^{3/2}, n^2\})$). In this paper, we bridge the gap between these closely related problems. More formally, we prove the following result.

► **Theorem 1.** *There exists an incremental algorithm for maintaining strongly connected components in directed graphs with expected total time $\tilde{O}(m^{4/3})$, where m refers to the number of edges in the final graph. The algorithm can also maintain the topological order of these components.*

Summary of Techniques. We obtain our results by extending the technique of [10] to the case of general directed graphs. Both [8] and [10] detect cycles by doing a graph search after the insertion of an edge (u, v) . However, they reduce their search space by only exploring “equivalent” vertices: vertices whose ancestor and descendant sets agree on a random subset S of V . A key ingredient of the analysis is a structural theorem of [8] that bounds the total number of equivalent pairs created by the sequence of insertions. However, their notion of equivalent vertices only applies to acyclic directed graphs. Additionally, the proof of this structural theorem (Lemma 3.2 and 3.5 of [8]) is rather unintuitive.

Our contributions are three-fold. We present a new proof of the structural theorem of [8], which is significantly simpler and more intuitive. We also show a natural generalization of this theorem to general directed graphs. Finally, we show how the framework of [10] can be extended to maintain SCCs in general graphs, rather than just doing cycle detection and topological sort in a DAG.

Related Problems. A closely related problem that has received a lot of attention is maintaining strongly connected components in a *decremental* graph. This problem has been widely studied (see e.g. [22, 19, 11, 9]) and a recent algorithm achieves near-optimal $\tilde{O}(m)$

■ **Table 1** Known Results for Incremental Cycle Detection, Topological Sort and SCC.

Reference	Update Time	Incremental SCC
[18]	$O(\min\{m^{3/2} \log n, m^{3/2} + n^2 \log n\})$	No
[20]	$O(m^{3/2} + m\sqrt{n} \log n)$	No
[2]	$O(n^{2.75})$	No
[6]	$\tilde{O}(n^2)$	No
[16]	$O(m^{3/2})$	Yes
[7]	$O(m \cdot \min\{m^{1/2}, n^{2/3}\}), \tilde{O}(n^2)$	Yes
[8]	$\tilde{O}(m\sqrt{n})$	No
[10]	$\tilde{O}(m^{4/3})$	No

total expected update time [9]. Although the goal in both problems is to maintain SCCs, the incremental and decremental versions have little overlap in terms of techniques. Another related problem is that of maintaining single-source shortest paths in an incremental directed graph. The current state-of-the-art for this problem is $\tilde{O}(n^2)$ in dense graphs [15] and $\tilde{O}(m\sqrt{n} + m^{7/5})$ in sparse ones [12].

2 Preliminaries

We consider the problem of maintaining strongly connected components in directed graphs in the incremental setting. In this setting, we start with an empty graph, and directed edges are added to the graph one at a time. We will let G refer to the current version of the graph, and its vertex and edge sets are denoted as V and E respectively. We use m to denote the total number of edges added to G and n to denote $|V(G)|$.

Consider two vertices $u, v \in V$. We say that the vertex u is an *ancestor* of v , and v is a *descendant* of u if there is a path from u to v in G . We will say that u and v are *related* if one is the ancestor of other. For $u \in V$, we use $A(u)$ and $D(u)$ to denote the current set of ancestors and descendants of u . Consider any $S \subseteq V$, for $u \in V$, we use $A_S(u)$ to denote the set $A(u) \cap S$, and $D_S(u)$ to denote the set $D(u) \cap S$. For any $v \in V$, we will use $C(v)$ to denote the strongly connected component containing v in the current graph G , and $|C(v)|$ will be the number of vertices contained in the component.

We will also use the following result due to Italiano [17] on single-source incremental reachability.

► **Lemma 2** ([17]). *Given $v \in V$, there exists an algorithm that maintains $A(v)$ and $D(v)$ in $O(m)$ total time during the course of insertion of m edges.*

We also use the following simplifying assumption by [8] (proved in the appendix of their paper).

► **Lemma 3** ([8]). *We can assume that every vertex in the current graph $G = (V, E)$ has degree $O(m/n)$.*

Data Structures Used. To maintain the strongly connected components, we use the disjoint set data structure of Tarjan [23]. This data structure stores the partition of the vertex set into disjoint sets. In our case, these disjoint sets will be the strongly connected components. Moreover, the disjoint sets are represented by a *canonical element*, which in this case will be a vertex. Following operations are supported by this data structure.

1. **FIND**(x): Given a vertex x , return the canonical vertex of the component containing x .
2. **LINK**(x, y): This operation joins the components whose canonical vertices are x and y . The newly formed component's canonical vertex is x .

This data structure supports any sequence of **FIND** and **LINK** operations in $O(n \log n)$ total time plus $O(1)$ time per operation. Our search and reordering operations will take $\Omega(n \log n)$ total time, so we can think of the **FIND** and **LINK** operations as being performed in $O(1)$ amortized time per operation.

Additionally, to maintain the topological ordering of the strongly connected component, we use the *ordered list* data structure of [14] and [5], which supports the following operations in $O(1)$ -time.

1. **INSERT-BEFORE**(x, y): This operation inserts the vertex x before the vertex y in the ordered list.
2. **INSERT-AFTER**(x, y): This operation inserts the vertex x after the vertex y in the ordered list.
3. **DELETE**(x): This operation deletes the vertex x from the current ordered list.
4. **ORDER**(x, y): This operation returns whether x appears before y in the ordering or not.

This data structure maintains the topological sort k of the strongly connected components implicitly. We will use some additional data structures for our algorithm, that we will mention when we discuss the algorithm.

3 Similarity

3.1 Previous Work

To bound the running time of their algorithm [8] introduced the notion of *sometime- τ -similar* pairs. We briefly discuss their definition.

► **Definition 4** ([8]). *A pair of vertices u and v are said to be sometime- τ -similar if there is a time t at which u is an ancestor of v , $|A(u) \oplus A(v)| \leq \tau$, and $|D(u) \oplus D(v)| \leq \tau$.*

The total number of *sometime- τ -similar* pairs are $\tilde{O}(n\tau)$. Note that this bound is false if we apply the same definition of similarity to the case of directed graphs with cycles. As an example, consider the case where the entire graph is a cycle. For such a graph, by Definition 4, we have $O(n^2)$ *sometime- τ -similar* pairs. So, a new definition of similarity is needed. Moreover, their proof strategy also uses the final topological ordering of the graph. Such an ordering is not possible in directed graphs with cycles. We overcome this by defining another ordering that (like topological ordering) is consistent with the incremental updates to the graph, but at the same time allows for strongly connected components.

3.2 A New Notion Of Similarity

► **Definition 5.** *Consider $u, v \in V$. Let $C(u)$ and $C(v)$ denote the strongly connected components containing u and v respectively, then u and v are called τ -similar in the current graph G if u and v are related, $|C(u)| \leq \tau$, $|C(v)| \leq \tau$, and $|A(u) \oplus A(v)| \leq \tau$, $|D(u) \oplus D(v)| \leq \tau$. Vertices u and v are called sometime- τ -similar, if they are τ -similar at some point during the course of m edge insertions.*

► **Remark 6.** Consider any $u, v \in V$ with $C(u) = C(v)$. If $|C(u)| \geq \tau + 1$ then u and v are not τ -similar in G . But if $|C(u)| \leq \tau$ then they are τ -similar.

With this remark, we distinguish between two types *sometime- τ -similar* vertices.

► **Definition 7.** We call u and v *related-sometime- τ -similar* if there is a time t when u and v are τ -similar with $C(u) \neq C(v)$. On the other hand if there is a time t when u and v are τ -similar and $C(u) = C(v)$, then we call u and v *equivalent-sometime- τ -similar*. It is possible for u, v to be both *related-sometime- τ -similar* and *equivalent-sometime- τ -similar*.

We show that the total number of *sometime- τ -similar* pairs are bounded.

► **Theorem 8.** The total number of *sometime- τ -similar* pairs are $\tilde{O}(n\tau)$.

Our proof will bound *related-sometime- τ -similar* pairs. It is easy to see that the number of *equivalent-sometime- τ -similar* is $O(n\tau)$.

► **Observation 9.** A vertex v can only be *equivalent-sometime- τ -similar* to the first τ vertices that join the same component as v . Thus, the total number of *equivalent-sometime- τ -similar* pairs is $O(n\tau)$.

To prove Theorem 8 we need the following claim.

▷ **Claim 10.** There exists a fixed total order I on the vertices of G which satisfies the following property:

1. Consider any $u, v \in V$. Let t_1 be the first time u and v become related such that u is an ancestor of v , then $I(u) < I(v)$.

Note that if the final graph G_m is acyclic, then I is satisfied by the topological ordering. We will show that it is possible to obtain an ordering that satisfies the above properties even if the graph has a cycle.

3.3 Existence of A Fixed Total Order

In this subsection, we define an ordering I that satisfies Claim 10.

► **Definition 11.** We define a relation \prec over the vertices of G : $u \prec v$ if and only if at some time t , u is an ancestor of v and $C(u) \neq C(v)$.

We first note that \prec is a strict partial order. We formally state and prove the following claim.

▷ **Claim 12.** The relation \prec on the vertices of G is a strict partial order.

Proof. We need to show that \prec is anti-symmetric and transitive. Anti-symmetry follows from the fact that for each pair of vertices u and v , either $u \not\prec v$ or $v \not\prec u$. Now suppose $u \prec v$ and $v \prec w$. Let t_1 be the time at which u is an ancestor of v and $C(u) \neq C(v)$. Similarly, let t_2 be the time at which v is an ancestor of w and $C(v) \neq C(w)$. Without loss of generality, assume that $t_2 \geq t_1$. Observe that u is an ancestor of w at time t_2 . If $C(u) = C(w)$, then $v \in C(w)$ at time t_2 as well, which is a contradiction. So, at time t_2 , $C(u) \neq C(w)$. This proves our claim. ◁

► **Definition 13.** We define I to be a linear extension of \prec . That is I is a total order consistent with \prec : if $u \prec v$, then $I(u) < I(v)$.

Proof of Claim 10. We claim that I of Definition 13 satisfies Claim 10. Consider any two vertices u and v , and let t_1 be the time at which u and v first become related, with u being an ancestor of v . Therefore at time t_1 , $C(u) \neq C(v)$, which implies that $u \prec v$. Since I is consistent with \prec , we know that $I(u) < I(v)$. ◁

3.4 Bounding the Number of Similar Pairs

In this section we will prove Theorem 8. From Observation 9, we conclude that it is sufficient to show that the number of *related-sometime- τ -similar* pairs are at most $O(n\tau \log n)$. We first introduce some notation. Moving forward we will use I to denote an ordering that satisfies Claim 10. We note that Theorem 8 can be obtained by combining the ordering I satisfying Claim 10 with a modification of the proof of *sometime- τ -similar* pairs in a DAG in Section 3 of [8]. However, even for the simpler case of DAGs, the proof in [8] requires a long case analysis. In this paper we present a different approach to the proof we believe is significantly simpler and more intuitive.

► **Definition 14.** Let u and v be a pair of *related-sometime- τ -similar* vertices. We denote it using an ordered tuple (u, v) if $I(u) < I(v)$.

► **Definition 15.** For a vertex v , we define $A^i(v)$ to be the set of vertices u such that (u, v) is a *related-sometime- τ -similar* pair, and $I(v) - I(u) \in [2^i, 2^{i+1})$. Similarly, we define $D^i(v)$ to be the set of vertices w such that (v, w) is a *related-sometime- τ -similar* pair, and $I(w) - I(v) \in [2^i, 2^{i+1})$.

► **Definition 16.** For a vertex v and a fixed i , we define the graph $G_v^{D,i}$ with the vertex set $D^i(v)$ and the graph $G_v^{A,i}$ with the vertex set $A^i(v)$ as follows.

1. Let $u_1, u_2, \dots, u_\alpha$ be the vertices of $A^i(v)$, where the vertices are ordered according to the increasing order of the time at which they become *related- τ -similar* with v . For $j < k$, we add an edge from u_j to u_k , if u_j is an ancestor of u_k when u_k first becomes τ -similar to v .
2. Let w_1, w_2, \dots, w_β be the vertices of $D^i(v)$, where the vertices are ordered according to the increasing order of time at which they become *related- τ -similar* with v . For $j < k$, we add an edge from w_j to w_k if w_j is a descendant of w_k when w_k first becomes τ -similar to v .

See Figure 1 for an illustration.

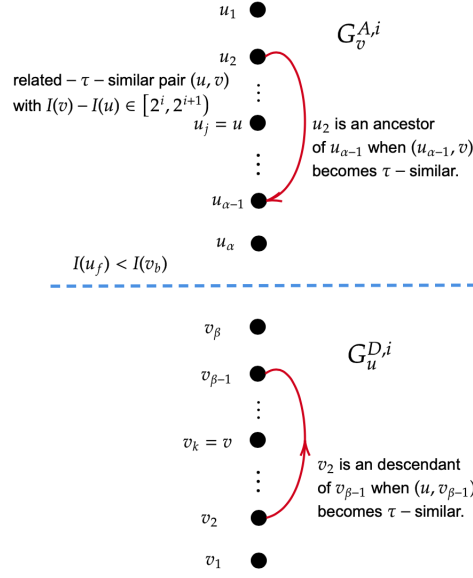
► **Claim 17.** Let (u, v) be a *related- τ -similar* pair such that $I(v) - I(u) \in [2^i, 2^{i+1})$. Consider $w \in A^i(v)$ and $z \in D^i(u)$, then $I(w) < I(z)$.

Proof. Suppose $I(z) < I(w)$. Note that $I(u) < I(z)$, and $I(w) < I(v)$. Consequently, $I(u) < I(z) < I(w) < I(v)$. Since $I(z) - I(u) \geq 2^i$, and $I(v) - I(w) \geq 2^i$, this implies that $I(v) - I(u) \geq 2^{i+1}$, which contradicts our assumption that $I(v) - I(u) \in [2^i, 2^{i+1})$. ◁

► **Claim 18.** For a vertex v , consider any $A^i(v) = \{u_1, \dots, u_\alpha\}$, where u_j are ordered in the increasing order of time at which they become *related- τ -similar* to v . Then the number of edges in $G_v^{A,i}$ coming into u_j is at least $j - \tau$. Similarly, let $D^i(v) = \{w_1, \dots, w_\beta\}$, where the vertices are ordered in the increasing order of time at which they become *related- τ -similar* with v . Then the number of edges coming into w_j in $G_v^{D,i}$ is at least $j - \tau$.

Proof. Let t be the time at which (u_j, v) become *related- τ -similar*. By t , for all $i < j$, (u_i, v) are *related- τ -similar*. If the in-degree of u_j is at most $j - \tau - 1$, then this implies that there are at least $\tau + 1$ vertices u_i , $i < j$ such that u_i is not an ancestor of u_j . However, these are all ancestors of v at time t . This implies that $|A(u_j) \oplus A(v)| \geq \tau + 1$, contradicting the fact that u_j and v are *related- τ -similar* at time t . ◁

► **Definition 19.** For a vertex v , consider $w \in A^i(v)$. We call w *bad with respect to v* if the outdegree of w in $G_v^{A,i}$ is at most 2τ . Similarly, we call a vertex $z \in D^i(v)$ *bad with respect to v* if the outdegree of z in $G_v^{D,i}$ is at most 2τ .



■ **Figure 1** We consider a *related- τ -similar* pair (u, v) , where $I(v) - I(u) \in [2^i, 2^{i+1})$. All vertices of $A^i(v)$ appear before the vertices of $D^i(u)$.

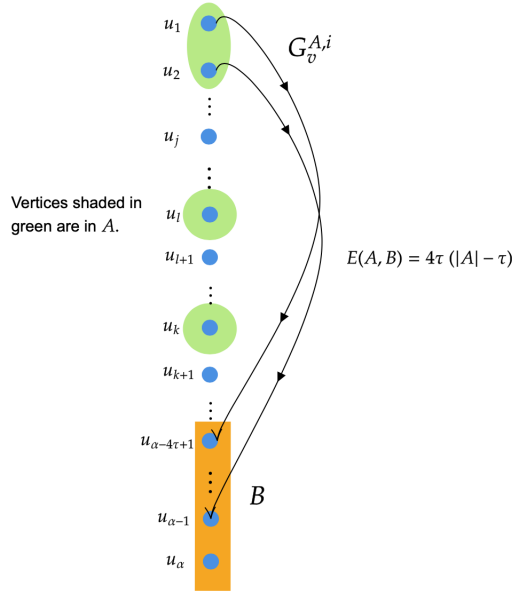
▷ **Claim 20.** For any v , the total number of bad vertices in $A^i(v)$ for any i is at most 6τ . Similarly, the total number of bad vertices in $D^i(v)$ for any i is at most 6τ .

Proof. As before, let $A^i(v) = \{u_1, u_2, \dots, u_{\alpha}\}$. Let $B = \{u_{\alpha-4\tau+1}, \dots, u_{\alpha}\}$. Let $A \subset A^i(v) \setminus B$ be the set of vertices outside of B that are bad for v (see Figure 2 for an illustration). We want to prove that $|A| \leq 2\tau$. This will give us the desired bound. Consider any $w \in B$. There are at least $|A| - \tau$ edges from A to w . So, the total number of edges going from A to B is at least $4\tau(|A| - \tau)$. The average outdegree of the vertices in A is at least $\frac{4\tau(|A| - \tau)}{|A|}$. Since the vertices in A are bad, we know that $\frac{4\tau(|A| - \tau)}{|A|} \leq 2\tau$. This implies that $|A| \leq 2\tau$. The proof for $D^i(v)$ is analogous. ◁

► **Lemma 21.** Let (u, v) be a *related-sometime- τ -similar* pair. Then, either u is bad for v or v is bad for u .

Proof. Let $I(v) - I(u) \in [2^i, 2^{i+1})$. As before we consider $A^i(v) = \{u_1, \dots, u_{\alpha}\}$, and let $D^i(u) = \{v_1, \dots, v_{\beta}\}$. Assume that neither u is bad for $A^i(v)$ nor v is bad for $D^i(u)$. This implies that the number of edges going out of u and v in $G_v^{A,i}$ and $G_u^{D,i}$, respectively, are at least $2\tau + 1$. Consider the *related- τ -similar* pairs $(u_1, v), \dots, (u_{\alpha}, v)$ and $(u, v_1), \dots, (u, v_{\beta})$. Note that among these pairs one of (u_{α}, v) or (u, v_{β}) are the last to become *related- τ -similar*. Without loss of generality, assume it is (u, v_{β}) . Since we assume that u is not bad with respect to v , at the point when (u_{α}, v) becomes *related- τ -similar*, u is an ancestor of at least $2\tau + 1$ vertices in u_1, \dots, u_{α} . Note that this claim also holds at the (later) time when (u, v_{β}) become *related-sometime- τ -similar*. We call this set of vertices U . We now consider two different cases:

1. If v_{β} is not an ancestor of at least $\tau + 1$ vertices in U , then this contradicts the fact that (u, v_{β}) is a *related- τ -similar* pair.
2. Suppose v_{β} is an ancestor of at least $\tau + 1$ vertices in U . Consider any $u_j \in U$. Observe from Claim 17 that $I(u_j) < I(v_{\beta})$. Since I satisfies Claim 10, we deduce that if v_{β} is an



■ **Figure 2** The vertices in green are the vertices of $A^i(v) \setminus B$ that are bad for v . Since the vertices in B are τ -similar to v , the total number of edges coming out of A and going into B is at least $4\tau(|A| - \tau)$.

ancestor of u_j , then it lies in the same strongly connected component as u_j . Since this is true for all $u_j \in U$, it follows that $|C(v_\beta)| \geq \tau + 1$, thus contradicting the fact that (u, v_β) is *related- τ -similar* (see Definition 5). ◀

► **Remark 22.** Observe that when we are in the acyclic case, then we don't have to deal with the second case at all, since I corresponds to the topological ordering of the final graph G_m .

Proof of Theorem 8. Consider a *related- τ -similar* pair (u, v) . We charge this pair to u if v is bad for u , and we charge it to v if u is bad for v . From Lemma 21, we know that each pair (u, v) is charged to either u or v . Finally, we observe that for any u , for a fixed i , the total number of bad vertices in $D^i(u)$ or $A^i(u)$ is at most 6τ each. Therefore, the total charge on each vertex is at most $12\tau \log n$ (since i is at most $\log n$). Since the total number of vertices is n , we know that the total charge, and therefore the total number of *related-sometime- τ -similar* pairs is at most $O(n\tau \log n)$. ◀

4 Equivalence

Consider $u, v \in V$, observe that u and v lie in the same strongly connected component iff $A(u) = A(v)$ and $D(u) = D(v)$. However, the sets $A(u)$ and $D(v)$ are expensive to maintain for all vertices. Therefore, Bernstein and Chechik [8] defined a relaxed notion of equivalence between vertices. We define a slightly different version that will be useful for our algorithm.

► **Definition 23.** (*S-equivalence*) Consider S which is created by including every vertex $v \in V$ independently with probability $12 \cdot \log n / \tau$, where τ is a parameter to be defined by the algorithm. Vertices u and v are called *S-equivalent* if they are related, $A_S(u) = A_S(v)$, and $D_S(u) = D_S(v)$. For the analysis of our algorithm, it will be useful to distinguish between two types of *S-equivalence*.

1. Vertices x and y are Type 1 if they satisfy the above-mentioned condition and $|C(x)| \geq \tau+1$ or $|C(y)| \geq \tau+1$.
2. Vertices x and y are Type 2 if they satisfy the above-mentioned condition and $|C(x)| \leq \tau$ and $|C(y)| \leq \tau$.

In [8], the algorithm samples a set S , and maintains a partition $\{V_{i,j}\}$ of V , where $V_{i,j} = \{u \in V \text{ s.t. } |A_S(u)| = i, |D_S(u)| = j\}$. We define an ordering \prec^* on these parts.

► **Definition 24.** We say that $V_{i,j} \prec^* V_{k,l}$ if either $\{i < k\}$, or $\{i = k \text{ and } j > l\}$ (note the slightly unusual ordering, instead of $j < l$, we have $j > l$). For $x \in V$, we use $V(x)$ to denote partition $V_{i,j}$ that contains x .

This partition has the following properties which were proved in [8] for directed acyclic graphs, but extend to general directed graphs as well.

► **Lemma 25.** Let $\{V_{i,j}\}$ be the partition of V maintained by the algorithm determined by the sampled set S , then

1. If x and y are related, with x being an ancestor of y , then either $V(x) \prec^* V(y)$ or $V(x) = V(y)$.
2. Consider a strongly connected component C of the current graph G , then $C \subseteq V_{i,j}$ for some i, j .
3. If x and y are related, and $V(x) = V(y)$, then x and y are S -equivalent.

Proof. We give a short proof of this lemma. If x is an ancestor of y , then $A(x) \subseteq A(y)$, and $D(y) \subseteq D(x)$. In particular, $A_S(x) \subseteq A_S(y)$ and $D_S(y) \subseteq D_S(x)$. This immediately tells us that $|A_S(x)| \leq |A_S(y)|$, and $|D_S(y)| \leq |D_S(x)|$ and the first part of the claim follows. Finally, consider any strongly connected component C , then for any $x, y \in C$, $A(x) = A(y)$, $D(x) = D(y)$. This implies that $A_S(x) = A_S(y)$, $D_S(x) = D_S(y)$ and this proves the second part of the claim. To see the third part, assume without loss of generality that x is an ancestor of y , and $V(x) = V(y)$. Note that $A_S(x) \subseteq A_S(y)$, and since $|A_S(x)| = |A_S(y)|$, we can conclude that $A_S(x) = A_S(y)$. Similarly, we can deduce that $D_S(x) = D_S(y)$, thus proving that x and y are S -equivalent. ◀

Keeping in mind Lemma 25, for a component C , we define $V(C)$ as the partition $V_{i,j}$ containing C . An important component of our algorithm is maintaining a topological sort k of the strongly connected components. This topological sort k will be consistent with the order \prec^* of the partitions. That is, for strongly connected components C and C' with $V(C) \prec^* V(C')$, $k(C) < k(C')$. The existence of such a topological ordering is guaranteed by Lemma 25. We will maintain a topological sort of the components by maintaining an ordered list on the canonical vertices. The components are disjoint and each of them have a unique canonical vertex. So, we will often use $k(\cdot)$ on canonical vertices as well.

The algorithms in [8] and [10] proceed by exploiting the notion of S -equivalence. This notion enables them reduce the space of vertices that need to be explored to detect cycles (from Lemma 25). Finally, they show that with high probability the total number of S -equivalent pairs is bounded, and the runtime of the algorithm is proportional to this number. In order to prove this claim, they show that S -equivalent pairs and sometime- τ -similar pairs are related. We show that our notion of sometime- τ -similarity can be used to bound the number of Type 2 S -equivalent pairs, instead of all S -equivalent pairs. It will be clear as we move forward why this is sufficient.

Recall Definition 23 and Definition 5. We show the following lemma relating *sometime- τ -similar* pairs and *Type 2 S -equivalent* pairs.

► **Lemma 26.** *Suppose $S \subseteq V$ is obtained by including each $x \in V$ independently with probability $\frac{12 \cdot \log n}{\tau}$. Suppose u and v are Type 2 S -equivalent, then with high probability, they are sometime- τ -similar.*

Observe that Theorem 8 and Lemma 26 together imply the following theorem:

► **Theorem 27.** *Let S be sampled by including $v \in V$ independently with probability $\frac{12 \cdot \log n}{\tau}$. Then, the total number of Type 2 S -equivalent pairs is at most $\tilde{O}(n\tau)$ with high probability.*

We now proceed to prove Lemma 26.

Proof of Lemma 26. Note that by the statement of the lemma, u and v are related, $|C(u)| \leq \tau$, $|C(v)| \leq \tau$. We additionally want to show that $|D(u) \oplus D(v)| \leq \tau$ and $|A(u) \oplus A(v)| \leq \tau$. Without loss of generality, assume that $|A(u) \oplus A(v)| \geq \tau + 1$. Then, applying Chernoff bound, we conclude that with probability at least $1 - O(1/n^5)$ there is a vertex $x \in A(u) \oplus A(v)$ that is included in S as well. This implies that u and v are not Type 2 S -equivalent. Taking union bound over all Type 2 S -equivalent pairs, which are at most n^2 in number, we conclude that with probability at least $1 - O(1/n^3)$ any Type 2 S -equivalent pair is also sometime- τ -similar. ◀

5 The Algorithm

When an edge (u, v) is inserted, the algorithm updates the newly formed strongly connected components, if any. Additionally, the algorithm maintains a topological sort k of the strongly connected components. This will be achieved by using canonical vertices as a proxy for the strongly connected components (see Section 2). These canonical vertices will be maintained as an ordered list, and when we are required to reorder the strongly connected components, the corresponding canonical vertices will be reordered. To achieve this, we follow the basic framework of [10]. The algorithm to process the insertion of (u, v) proceeds in the following phases.

1. **Phase 1.** This phase is responsible for maintaining reachability information to and from S (using Lemma 2). Additionally, in this phase, the algorithm uses this reachability information to update the sets $V_{i,j}$ and to handle the case where the new SCC formed by the insertion of (u, v) contains at least one vertex in S . If the algorithm finds such an SCC, it terminates after Phase 1, i.e. it skips Phases 2 and 3.
2. **Phase 2.** This phase is responsible for handling small SCCs. In particular, it detects the case when (u, v) creates a new SCC that does not contain any $s \in S$, as well as the case where (u, v) creates no new SCC. The phase also links together the canonical vertices corresponding to this new SCC (if any).
3. **Phase 3.** This phase updates the topological order of the strongly connected components by reordering canonical vertices. Note that even if (u, v) creates no SCCs, Phase 3 may need to do some reordering to ensure that k remains a valid topological order.

In the main body of the paper, we will describe Phases 2 and 3 and the subroutines used in these phases. The correctness of these subroutines can be found in the full version of the paper. Phase 1 is essentially the same as in the framework of [8], so we postpone the details to the full version of the paper.

5.1 Phase 1: Updating the partition $\{V_{i,j}\}$ and Handling Large SCCs

In this section, we give an overview of Phase 1 and its guarantees. The detailed description is in the full version of the paper. Using Lemma 2, we can maintain reachability to and from every vertex in S in total time $O(m|S|)$ over all edge insertions. This allows us to maintain two additional piece of information. Recall the partition $V_{i,j}$ from Section 4, and note that every $V(x)$ is determined entirely by $A_S(x)$ and $D_S(x)$. Thus, Phase 1 can use the reachability information to/from S to maintain the partition $V_{i,j}$. Phase 1 can also use this reachability information to detect any new SCCs that contain a vertex in S .

We now state these guarantees more formally. The following lemma is essentially identical to the guarantees of [8], but is modified to handle SCCs.

► **Lemma 28** ([8]). *Consider the insertion of edge (u, v) . Phase 1 has the following guarantees:*

1. *At the end of Phase 1, each set $V_{i,j}$ is correct for the new version of the graph (the graph with edge (u, v) inserted). The algorithm also updates the order of the strongly connected components so that they are consistent with \prec^* .*
2. *If the insertion of (u, v) creates a new SCC that contains a vertex in S , then Phase 1 detects the new SCC, links the corresponding canonical vertices, and computes the topological order of the resulting SCCs. The update procedure then terminates and does not continue to Phase 2 or 3.*
3. *If the insertion of (u, v) does not create a new SCC that contains a vertex in S , then Phase 1 does not create any new SCCs. In this case, after the end of Phase 1, the ordering k on the canonical vertices is guaranteed to be a valid topological ordering of the canonical vertices in $G \setminus \{(u, v)\}$. The algorithm then proceeds to Phases 2 and 3.*

5.2 Phase 2: Detecting Small SCCs

The algorithm enters Phase 2 only if the newly inserted edge (u, v) does not create a new SCC that contains a vertex of S ; otherwise the algorithm to process (u, v) terminates after Phase 1. We also remark that if the algorithm enters Phase 2, then with high probability the size of the newly formed strongly connected component (if one exists) is at most τ . This follows from an easy application of Chernoff bound: if the newly formed component has size at least $\tau + 1$, then with high probability, it contains a vertex of S , in which case the algorithm terminates after Phase 1. Taking a union bound over all n^2 edge insertions, we get the following:

► **Observation 29.** *If the algorithm enters Phase 2 while processing an edge (u, v) , then with high probability, the new strongly connected components formed by the addition of (u, v) (if one exists) has size at most τ .*

Additionally, recall that Phase 1 updates the partition set $\{V_{i,j}\}$, so we assume that once we enter Phase 2 this partition already corresponds to the graph G (Lemma 28).

Previous Work. Our Phase 2 will be similar to the cycle detection algorithm of [10], but we need to adapt it to find the newly formed strongly connected component. Previous algorithms for finding SCCs such as the one by [16], proceed by implementing the cycle detection algorithm, but running it only over the canonical vertices. However, our algorithm will do a search over all vertices of the graph. We do this because sizes of the SCCs will be relevant to the runtime of the algorithm, and they weren't relevant in the case of [16].

We now give a brief outline of Phase 2: when an edge (u, v) is added to the graph, then the algorithm first checks if $k(\text{FIND}(u)) < k(\text{FIND}(v))$. If this is the case, then there couldn't have been an existing path from v to u (due to Lemma 28). As a result, a new component

containing u and v could not be formed. So, the algorithm doesn't continue. To detect if a new component is formed and to find all the vertices of this component, the algorithm does alternate steps of forward and backward search. For this purpose, it maintains sets F_a and F_d (to do forward search), B_a and B_d (to do backward search). For the forward search, F_a and F_d are the vertices that are alive (yet to be explored), and dead (already explored). Sets B_a and B_d are similarly defined for the backward search. When we encounter a vertex while exploring in the forward direction, we add it to F_a . When all neighbors of a vertex $v \in F_a$ that are S -equivalent to v been added to $F_a \cup F_d$, we add v to F_d . We add vertices to B_a and B_d similarly. At all times, while exploring vertices in the forward direction, we want to stay as close to v as possible, so we pick out a vertex x with minimum $k(\text{FIND}(x))$ from F_a to explore next. Similarly, while exploring in the backward direction, we want to stay as close to u as possible, so we pick out the vertex y with maximum $k(\text{FIND}(y))$ from B_a to explore next in the backward direction. We refer the reader to Algorithms 1, 2, and 4 for pseudocodes for Phase 2. We give the proof of correctness in the full version of the paper. We briefly outline the proof of runtime.

► **Lemma 30.** *The total runtime of Phase 2 is $O(\sqrt{m^3\tau/n})$.*

Proof Sketch. Suppose we have process edge e_t and let f_t denote the size of F_d after $\text{FINDCOMPONENT}()$ has finished terminated. We observe that $|B_d| = \Theta(f_t)$ as well, since we do a balanced search. From Lemma 3 we conclude that the total update time of the algorithm over m edge insertions is $O(m/n \sum_{t=1}^m f_t)$. The goal is to now bound $\sum_{t=1}^m f_t$. Consider $x \in F_d$ and $y \in B_d$ after $\text{FINDCOMPONENT}()$ has finished processing e_t . We show that (x, y) is a newly formed *related- τ -similar* pair or *equivalent- τ -similar* pair. This implies that $\sum_{t=1}^m f_t^2 = \tilde{O}(n\tau)$ (from Theorem 8). Using Cauchy-Schwarz, we know that $\sum_{t=1}^m f_t = \tilde{O}(\sqrt{mn\tau})$. Thus the total runtime of Phase 2 is $O(\sqrt{m^3\tau/n})$. ◀

5.3 Phase 3: Sorting the Canonical Vertices

We enter this Phase only if there is no vertex of S in the newly created SCC, C_N . After Phase 1 and Phase 2, we know which canonical vertices have combined to give the newly formed strongly connected component. We delete these canonical vertices from the ordered list, and show how to reorder the list so that a topological sort on the canonical vertices is maintained.

To update the topological ordering of the canonical vertices, we follow the framework of [10]. We present it here for completeness, modifying their algorithm slightly to account for the case where a cycle is created.

We will consider two cases, one where a new component is created and one where no new component is created. Suppose no new component is created, and consider the sets F_d and B_d , from the forward and backward searches after we have processed edge (u, v) . Since we do an ordered search, we know that all the vertices of a given component appear in a continuous manner in F_d and B_d . Let $\text{FIND}(v), x_1, \dots, x_f$ be the **canonical** vertices corresponding to the components appearing in F_d , with $k(\text{FIND}(v)) < k(x_1) < \dots < k(x_f)$. Similarly, let $y_1, y_2, \dots, y_b, \text{FIND}(u)$ be the **canonical** vertices corresponding to the components appearing in B_d , with $k(y_1) < k(y_2) < \dots < k(y_b) < k(\text{FIND}(u))$. We use the subroutine $\text{UPDATEFORWARD}()$ and $\text{UPDATEBACKWARD}()$ to update the ordered list (see Algorithm 3). This list only consists of canonical vertices that represent different components.

We now describe how to reorder the vertices. In [10] two cases are considered, the first case corresponds to when the algorithm terminates in conditions: $B_a = \emptyset$ or $\max_{x \in B_a} k(\text{FIND}(x)) < \max_{y \in F_d} k(\text{FIND}(y))$. For this case, we use the subroutine $\text{UPDATEFORWARD}()$. The proof

for the case when the algorithm terminates in conditions $F_a = \emptyset$ or $\min_{x \in F_a} k(\text{FIND}(x)) > \min_{y \in B_d} k(\text{FIND}(y))$ is analogous (we use a subroutine `UPDATEBACKWARD()`) and we omit it here. We postpone the proof of correctness to the full version. We give a proof of the runtime.

► **Lemma 31.** *The total runtime of Phase 3 is $O(\sqrt{mn\tau})$.*

Proof sketch. For each $x \in F_d$ and $y \in B_d$, the algorithm `UPDATEFORWARD()` puts `FIND(x)` and `FIND(y)` in the correct position in the ordered list. This takes time $O(1)$ per vertex in F_d and B_d , giving a total runtime of $O(\sqrt{mn\tau})$. ◀

When a new component C_N is formed. If a new strongly connected component C_N is formed, and it doesn't contain a vertex of S , then the algorithm still needs to reorder some components. Assume without loss of generality that v is the canonical vertex of C_N . We first proceed to delete from the ordered list, all canonical vertices corresponding to the components that combined to form C_N . We define $x_1, x_2, \dots, x_f \in F_d$ and $y_1, y_2, \dots, y_b \in B_d$ as before except we exclude the canonical vertices that combined to form C_N . Finally, if our `FINDCOMPONENT()` terminated in $B_a = \emptyset$ or $\max_{x \in B_a} k(\text{FIND}(x)) \leq \max_{y \in F_d} k(\text{FIND}(y))$, then we execute `UPDATEFORWARD()`, else if `FINDCOMPONENT()` terminated in $F_a = \emptyset$ or $\min_{x \in F_a} k(\text{FIND}(x)) \geq \min_{y \in B_d} k(\text{FIND}(y))$, then we execute `UPDATEBACKWARD()`. The proof of correctness can be found in the full version of the paper and is the same as in the case when there is no new strongly connected component formed.

► **Lemma 32.** *The total update time of our algorithm is $\tilde{O}(m^{4/3})$.*

Proof. The total time taken in Phase 1, 2 and 3 is at most $\tilde{O}(mn/\tau + \sqrt{mn\tau} + \sqrt{m^3\tau/n})$. Substituting $\tau = n/m^{1/3}$, we get the desired bound of $\tilde{O}(m^{4/3})$. ◀

■ **Algorithm 1** `EXPLORE-FORWARD(x)`.

```

1  $F_a = F_a \setminus \{x\}$  and  $F_d = F_d \cup \{x\}$ .
2 for  $x' \in \text{out}(x)$  with  $V(x) = V(x')$  do
3   if  $\text{FIND}(x') \in F_a \cup F_d$  then
4      $\text{CYCLE} = 1$ 
5   if  $x' \notin B_a \cup B_d$  then
6      $\text{add } x' \text{ to } B_a$ .
```

■ **Algorithm 2** `EXPLORE-BACKWARD(x)`.

```

1  $B_a = B_a \setminus \{x\}$  and  $B_d = B_d \cup \{x\}$ .
2 for  $x' \in \text{in}(x)$  with  $V(x) = V(x')$  do
3   if  $\text{FIND}(x') \in F_a \cup F_d$  then
4      $\text{CYCLE} = 1$ 
5   if  $x' \notin B_a \cup B_d$  then
6      $\text{add } x' \text{ to } B_a$ .
```

Algorithm 3 UPDATEFORWARD().

```

1  $Q = F_d.$ 
2  $x^* = \arg \max \{k(\text{FIND}(x)) \mid x \in Q, x \text{ canonical}\}.$ 
3  $Q = Q \setminus C(x^*).$  // Since we are rearranging canonical vertices.
4 while  $Q \neq \emptyset$  do
5    $x' = \arg \max_{x \in Q} \{k(\text{FIND}(x))\}.$ 
6    $Q = Q \setminus C(x').$ 
7   INSERT-BEFORE(FIND( $x'$ ),  $x^*$ )
8    $x^* = \text{FIND}(x').$ 
9  $y^* = \text{FIND}(v).$ 
10  $Q = B_d.$ 
11 while  $Q \neq \emptyset$  do
12    $y' = \arg \max_{y \in B_d} k(\text{FIND}(y)).$ 
13    $Q = Q \setminus C(y').$ 
14   INSERT-BEFORE(FIND( $y'$ ),  $y^*$ )
15    $y^* = \text{FIND}(y').$ 

```

References

- 1 Deepak Ajwani and Tobias Friedrich. Average-case analysis of incremental topological ordering. *Discrete Appl. Math.*, 158(4):240–250, 2010. doi:10.1016/j.dam.2009.07.006.
- 2 Deepak Ajwani, Tobias Friedrich, and Ulrich Meyer. An $o(n^{2.75})$ algorithm for incremental topological ordering. *ACM Trans. Algorithms*, 4(4), 2008. doi:10.1145/1383369.1383370.
- 3 Bowen Alpern, Roger Hoover, Barry K. Rosen, Peter F. Sweeney, and F. Kenneth Zadeck. Incremental evaluation of computational circuits. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 32–42, USA, 1990. Society for Industrial and Applied Mathematics.
- 4 Ferenc Belik. An efficient deadlock avoidance technique. *IEEE Trans. Comput.*, 39(7):882–888, 1990. doi:10.1109/12.55690.
- 5 Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, page 152–164, Berlin, Heidelberg, 2002. Springer-Verlag.
- 6 Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. A new approach to incremental topological ordering. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, page 1108–1115, USA, 2009. Society for Industrial and Applied Mathematics.
- 7 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22, 2016. doi:10.1145/2756553.
- 8 Aaron Bernstein and Shiri Chechik. Incremental topological sort and cycle detection in $\tilde{O}(m\sqrt{n})$ expected total time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–34. SIAM, 2018.
- 9 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23–26, 2019*, pages 365–376. ACM, 2019.
- 10 Sayan Bhattacharya and Janardhan Kulkarni. An improved algorithm for incremental cycle detection and topological ordering in sparse graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2509–2521. SIAM, 2020.

Algorithm 4 FINDCOMPONENT(u, v).

```

1 if  $k(\text{FIND}(u)) < k(\text{FIND}(v))$  then
2    $\perp$  return NO.
3 Initialize  $\text{CYCLE} = 0$  and min-heaps  $F_a = \{v\}, B_a = \{u\}, F_d = \emptyset, B_d = \emptyset$ .
4 if  $\text{FIND}(u) = \text{FIND}(v)$  or  $V(u) \neq V(v)$  then
5    $\perp$  return NO.
6 while  $F_a \neq \emptyset$  and  $B_a \neq \emptyset$  do
7   Let  $x = \arg \min_{x' \in F_a} k(\text{FIND}(x'))$ .
8   if  $k(\text{FIND}(x)) > \min_{z' \in B_d} k(\text{FIND}(z'))$  and  $\text{CYCLE} = 0$  then
9      $\perp$  EXIT LOOP.
10  if  $k(\text{FIND}(x)) > \min_{z' \in B_d} k(\text{FIND}(z'))$  and  $\text{CYCLE} = 1$  then
11     $\perp$  EXIT LOOP.
12  if  $k(\text{FIND}(x)) = \min_{z' \in B_d} k(\text{FIND}(z'))$  and  $\text{CYCLE} = 1$  then
13     $\perp$  EXIT LOOP.
14  else
15     $\perp$  set  $\text{STATUS}(x) = 1$  and EXPLORE-FORWARD( $x$ ).
16  Let  $y = \arg \max_{y' \in B_a} k(\text{FIND}(y'))$ .
17  if  $k(\text{FIND}(y)) < \max_{y' \in F_d} k(\text{FIND}(y'))$  and  $\text{CYCLE} = 0$  then
18     $\perp$  EXIT LOOP.
19  if  $k(\text{FIND}(y)) < \max_{y' \in F_d} k(\text{FIND}(y'))$  and  $\text{CYCLE} = 1$  then
20     $\perp$  EXIT LOOP.
21  if  $k(\text{FIND}(y)) = \max_{y' \in F_d} k(\text{FIND}(y'))$  and  $\text{CYCLE} = 1$  then
22     $\perp$  EXIT LOOP.
23  else
24     $\perp$  set  $\text{STATUS}(y) = 1$  and EXPLORE-BACKWARD( $y$ ).
25 if  $\text{CYCLE} = 0$  then
26    $\perp$  return NO
27 if  $\text{CYCLE} = 1$  then
28   if the algorithm ended in 12 (or 20) then
29     Let  $z^* = \arg \min_{z' \in B_d} k(\text{FIND}(z'))$  (or  $z^* = \arg \max_{z' \in F_d} k(\text{FIND}(z'))$ ).
30     Do a DFS backwards from  $u$ , over the set of vertices  $x$  with  $\text{STATUS}(x) = 1$ .
31     Mark those that reach some vertex in  $C(z^*)$  or  $C(v)$ .
32     Do a DFS forwards from  $v$ , over the set of vertices  $x$  with  $\text{STATUS}(x) = 1$ .
33     Mark those that reach some vertex in  $C(v)$  or  $C(z^*)$ .
34   if the algorithm ended in 10 (or 18) then
35     Do a forward DFS search from  $v$ , over the set of vertices  $x$  with
36      $\text{STATUS}(x) = 1$ , mark those that reach some vertex in  $C(u)$ .
37   Let  $z$  be a marked canonical vertex.
38   for all canonical  $x \neq z$  that is marked do
39      $\perp$  LINK( $z, x$ ).

```

- 11 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Lacki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $\tilde{O}(m\sqrt{n})$ total update time. 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 315–324. IEEE Computer Society, 2016.
- 12 Shiri Chechik and Tianyi Zhang. Incremental single source shortest paths in sparse digraphs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2463–2477. SIAM, 2021. doi:10.1137/1.9781611976465.146.
- 13 Edith Cohen, Amos Fiat, Haim Kaplan, and Liam Roditty. A labeling approach to incremental cycle detection. *CoRR*, abs/1310.8381, 2013. arXiv:1310.8381.
- 14 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 365–372, 1987.
- 15 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 153–166. ACM, 2020.
- 16 Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert E. Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms*, 8(1), 2012. doi:10.1145/2071379.2071382.
- 17 Giuseppe F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273–281, 1986.
- 18 Irit Katriel and Hans L. Bodlaender. Online topological ordering. *ACM Trans. Algorithms*, 2(3):364–379, 2006. doi:10.1145/1159892.1159896.
- 19 Jakub Lacki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms*, 9(3):27:1–27:15, 2013. doi:10.1145/2483699.2483707.
- 20 Hsiao-Fei Liu and Kun-Mao Chao. A tight analysis of the katriel-bodlaender algorithm for online topological ordering. *Theor. Comput. Sci.*, 389(1-2):182–189, 2007. doi:10.1016/j.tcs.2007.08.009.
- 21 Alberto Marchetti-Spaccamela, Umberto Nanni, and Hans Rohnert. Maintaining a topological order under edge insertions. *Inf. Process. Lett.*, 59(1):53–58, 1996. doi:10.1016/0020-0190(96)00075-0.
- 22 Liam Roditty. Decremental maintenance of strongly connected components. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1143–1150. SIAM, 2013.
- 23 Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

Lyndon Words Accelerate Suffix Sorting

Nico Bertram ✉

Department of Computer Science, Technische Universität Dortmund, Germany

Jonas Ellert ✉ 

Department of Computer Science, Technische Universität Dortmund, Germany

Johannes Fischer ✉

Department of Computer Science, Technische Universität Dortmund, Germany

Abstract

Suffix sorting is arguably the most fundamental building block in string algorithmics, like regular sorting in the broader field of algorithms. It is thus not surprising that the literature is full of algorithms for suffix sorting, in particular focusing on their practicality. However, the advances on practical suffix sorting stalled with the emergence of the DivSufSort algorithm more than 10 years ago, which, up to date, has remained the fastest suffix sorter. This article shows how properties of Lyndon words can be exploited algorithmically to accelerate suffix sorting again. Our new algorithm is 6–19% faster than DivSufSort on real-world texts, and up to three times as fast on artificial repetitive texts. It can also be parallelized, where similar speedups can be observed. Thus, we make the first advances in practical suffix sorting after more than a decade of standstill.

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

Keywords and phrases Suffix array, suffix sorting, Lyndon words, string algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.15

Supplementary Material *Software:* <https://github.com/jonas-ellert/gsaca-double-sort>

Software: <https://github.com/jonas-ellert/gsaca-lyndon>

Funding This work has been supported by the German Research Association (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project A6.

1 Introduction & Related Work

Since its introduction [13] in 1990, the suffix array – storing the order of the lexicographically sorted suffixes – has become one of the most important data structures in the field of string processing. Its applications include text indexing, text compression, and in particular the construction of the Burrows–Wheeler transformation.

From a theoretical point of view, the story is almost over: the suffix array can be computed in asymptotically optimal $\mathcal{O}(n)$ time and using only $\mathcal{O}(1)$ additional words of working space [9]¹. However, the fast *practical* construction of the suffix array remains an active topic of research. The efficiency of existing suffix sorters varies immensely [2], and the worst-case time and space bounds do not accurately predict the real world performance. In fact, the practically fastest and also highly memory efficient algorithm DivSufSort is not amongst the linear time algorithms [6], and has remained on the top of the scoreboard ever since its introduction more than a decade ago.

In 2016, Baier introduced the algorithm GSACA [4], which is the first to construct the suffix array in linear time without using recursion. It utilizes properties of so-called *Lyndon words*, which are strings that are lexicographically smaller than all of their proper

¹ for integer alphabet $[1, \sigma]$ with $\sigma \leq n$



© Nico Bertram, Jonas Ellert, and Johannes Fischer;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 15; pp. 15:1–15:13

Leibniz International Proceedings in Informatics



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

suffixes (for example, **artist** is a Lyndon word; **concert** is not a Lyndon word because it is lexicographically larger than its suffix **cert**). Conceptually, the algorithm consists of two phases. Franek et al. showed that the first phase computes (a partially sorted version of) the Lyndon array (a definition follows later), which is then used in the second phase to induce the suffix array [8]. Despite its interesting theoretical properties, GSACA cannot compete with the best suffix sorters in practice.

Our Contributions. We make the first advances on practical suffix sorting since the introduction of DivSufSort more than a decade ago. Our starting point is the GSACA algorithm, but we show how special properties of Lyndon words allow us to use fast integer sorting algorithms for its two phases. As a result, we obtain an efficient algorithm that is also easy to parallelize. Our sequential implementation is around 6–19% faster than DivSufSort on real-world inputs, and up to three times as fast on artificial repetitive inputs. However, it comes at the cost of a larger memory footprint (even though we still use much less space than the original implementation of GSACA). Our parallelization scales well up to at least 16 cores, and on large inputs it is faster than Labeit’s parallel implementation of DivSufSort [12], the currently fastest shared memory suffix sorter.

The rest of the paper is organized as follows: In Section 2 we introduce the definitions and notation that we use throughout the paper. We explain our new version of GSACA in Section 3, and give implementation details and a description of our parallelization in Section 4. We conclude the paper with an experimental evaluation in Section 5.

2 Preliminaries

We write $\lg x$ for $\log_2 x$. For $i, j \in \mathbb{N}$, we use the closed, half-open, and open interval notations $[i, j] = [i, j + 1) = (i - 1, j] = (i - 1, j + 1)$ to represent the set $\{x \mid x \in \mathbb{N} \wedge i \leq x \leq j\}$. Our analysis is performed in the word RAM model [10], where we can perform fundamental operations (logical shifts, basic arithmetic operations etc.) on words of size w bits in constant time. For the input size n of our problems we assume $\lceil \lg n \rceil \leq w$.

A *string* (also called *text*) over the *alphabet* Σ is a finite sequence of *symbols* from the finite and totally ordered set Σ . We say that a string S has length n and write $|S| = n$, if S is a sequence of exactly n symbols. The string of length 0 is called empty string and denoted by ϵ . The i -th symbol of a string S is denoted by $S[i]$, while the *substring* from the i -th to the j -th symbol is denoted by $S[i..j]$. For $i > j$ we define $S[i..j] = \epsilon$. For convenience, we use the interval notations $S[i..j + 1) = S(i - 1..j] = S(i - 1..j + 1) = S[i..j]$. The i -th *suffix* of S is defined as $S_i = S[i..n]$, while the substring $S[1..i]$ is called *prefix* of S . A prefix or suffix of S is called *proper*, if and only if its length is at least 1 and at most $n - 1$. Let S and T be two strings over Σ of lengths n and m , respectively. The concatenation of S and T is denoted by ST . The length of the *longest common prefix* (LCP) between S and T is defined as $\text{LCP}(S, T) = \max\{\ell \mid \ell \in [0, \min(n, m)] \wedge S[1..\ell] = T[1..\ell]\}$. The *longest common extension* (LCE) of indices i and j is the length of the LCP between S_i and S_j , i.e. $\text{LCE}(i, j) = \text{LCP}(S_i, S_j)$. The total order on Σ induces a total order on the set Σ^* of strings over Σ . Let S and T be strings over Σ , and let $\ell = \text{LCP}(S, T)$. We say that S is lexicographically smaller than T and write $S \prec T$, if and only if either $\ell = n < m$ (i.e. S is a prefix of T) or $\ell < \min(n, m) \wedge S[\ell + 1] < T[\ell + 1]$. We write $S \preceq T$ to denote $S \prec T \vee S = T$.

We can simplify the description of our algorithm with a special symbol $\$ \notin \Sigma$ that is smaller than all symbols from Σ . We say that S is *null-terminated*, if $S[n] = \$ \wedge \forall i \in [1, n) : S[i] \neq \$$.

Lexicographical Ordering of Suffixes. The *suffix array* lexicographically orders the suffixes of a string. To save space we only store the starting index of each suffix.

► **Definition 1** (Suffix Array). *Given a string S of length n , its suffix array A is the unique permutation of $[1, n]$ that satisfies $S_{A[1]} \prec S_{A[2]} \prec \dots \prec S_{A[n]}$. The inverse suffix array A^{-1} is the inverse permutation of A , i.e. $\forall i \in [1, n] : A^{-1}[A[i]] = i$.*

Baier's algorithm computes the suffix array by exploiting properties of *Lyndon words*. A Lyndon word is a string that is lexicographically smaller than all of its proper suffixes, i.e. S is a Lyndon word if and only if $\forall i \in [2..n] : S \prec S_i$ [5]. The Lyndon array of S identifies the longest Lyndon word starting at each text position.

► **Definition 2** (Lyndon Array). *Given a string S of length n , its Lyndon array λ is defined by $\forall i \in [1, n] : \lambda[i] = \max\{\ell \mid \ell \in [1, n - i + 1] \wedge S[i..i + \ell) \text{ is a Lyndon word}\}$. We write $w_\lambda(i) = S[i..i + \lambda[i]]$ to denote the longest Lyndon word that starts at index i .*

An important property of the Lyndon array is that it inherently encodes some information about the lexicographical ordering of the suffixes.

► **Lemma 3** ([7, Lemma 15]). *Let S be a string of length n with Lyndon array λ , and let $i \in [1, n]$. It holds $\lambda[i] = \ell$ if and only if*

$$(i + \ell \leq n + 1) \wedge (i + \ell \leq n \implies S_i \succ S_{i+\ell}) \wedge (\forall j \in (i, i + \ell) : S_i \prec S_j).$$

We conclude the preliminaries by showing two relations between Lyndon words and the lexicographical order of suffixes:

► **Lemma 4.** *Let S be a string of length n , let λ be its Lyndon array, and let $i, j \in [1, n]$ be arbitrary indices. If $w_\lambda(i) \prec w_\lambda(j)$, then $S_i \prec S_j$.*

Proof. If $w_\lambda(i) \prec w_\lambda(j)$, then $w_\lambda(j)$ is not a prefix of $w_\lambda(i)$. If also $w_\lambda(i)$ is not a prefix of $w_\lambda(j)$, then the first mismatch between $w_\lambda(i)$ and $w_\lambda(j)$ determines the lexicographical order of S_i and S_j . Thus we only have to consider the case where $w_\lambda(j) = w_\lambda(i)\alpha$ for a non-empty string α . Let $S[i..i + \lambda[j]] = w_\lambda(i)\beta$, then it must hold $\beta \prec \alpha$ and thus also $S_i \prec S_j$. Otherwise, [5, Prop. 1.5] would imply that $w_\lambda(i)\beta$ is a Lyndon word. ◀

► **Lemma 5.** *Let $S_i = \alpha S_j$ (with $j = i + |\alpha|$) be a suffix of a string, where α is a Lyndon word. It holds $S_i \succ S_j \iff w_\lambda(i) = \alpha$. If $S_i \prec S_j$, then $\alpha w_\lambda(j)$ is a Lyndon word.*

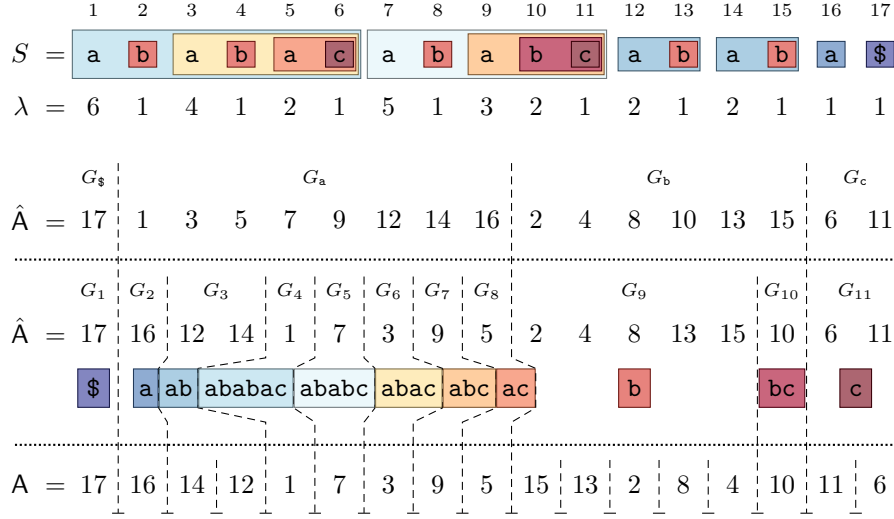
Proof. Lemma 3 directly implies $S_i \succ S_j \iff w_\lambda(i) = \alpha$. Assume $S_i \prec S_j$, then Lemma 4 implies $w_\lambda(j) \succeq \alpha$. If $w_\lambda(j) = \alpha$, then due to Lemma 3 it holds $S_j \succ S_{j+|\alpha|}$, which leads to the contradiction $S_i = \alpha S_j \succ \alpha S_{j+|\alpha|} = S_j$. Thus it holds $w_\lambda(j) \succ \alpha$, and [5, Prop. 1.3] implies that $\alpha w_\lambda(j)$ is a Lyndon word. ◀

3 Sequential Algorithm

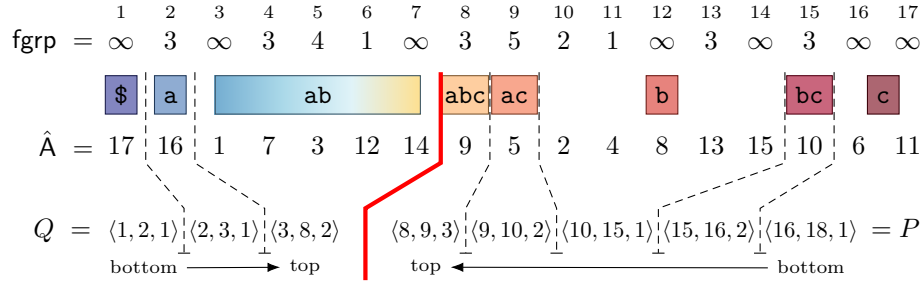
We start by giving a high level description of Baier's algorithm. For clarity, we write \hat{A} to denote the *not yet computed* suffix array, i.e. an array that serves as preliminary storage during the execution of the algorithm, and ultimately becomes the actual suffix array A . We use the terms suffix and index interchangeably. The algorithm consists of three main steps. For each step, we provide an example in Figure 1a.

Initialization: We sort and group the suffixes by their first symbol. The suffixes of each group are stored in increasing index order in a consecutive interval of \hat{A} , and the order of the intervals is determined by the rank of the starting symbols. In our example, the order of the alphabet is $\$ < a < b < c$. Thus, the leftmost group contains the suffixes that start with $\$$, followed by the group of suffixes that start with a , and so forth.

15:4 Lyndon Words Accelerate Suffix Sorting



(a) Baier's algorithm. Initially, we group the indices by symbol (above the first dotted line). In the first phase, we group the indices by longest Lyndon words (below the first dotted line). In the second phase, we lexicographically sort the suffixes within each group (below the second dotted line).



(b) Data structures during Phase 1. The stack Q contains groups that we may still have to refine (left of the red line). The stack P contains only final groups (right of the red line).

Figure 1 Baier's algorithm and data structures used during Phase 1. The colored boxes represent the group contexts, which are also exactly the longest Lyndon words in (a). (Best viewed in color.)

Phase 1: We refine the groups such that two suffixes S_i and S_j belong to the same group if and only if they share the longest Lyndon word $w_\lambda(i) = w_\lambda(j)$. Again, the indices of each group are stored in increasing order in a consecutive interval of \hat{A} . The order of the groups is determined by the lexicographical order of the Lyndon words. In our example it holds $w_\lambda(3) = \text{abac} \prec \text{abc} = w_\lambda(9)$, and thus the group containing index 3 is stored to the left of the group containing index 9. From Lemma 4 follows that the grouping after Phase 1 is compatible with the suffix array.

Phase 2: We lexicographically sort the suffixes within each group to obtain the suffix array.

Baier uses a special form of induced copying (see e.g. [11, 14]) for Phase 1 and 2, which is elegant but results in a noncompetitive practical performance (see [4, Table 2] and [3, Chapter 6]). In the remainder of this section we explain how to instead use integer sorting for these phases, which allows a more efficient implementation of the algorithm. First, we give a formal definition of the grouping structure.

► **Definition 6** (Suffix Grouping). A suffix grouping consists of an array \hat{A} and a list G_1, \dots, G_m of groups. A group with context $\alpha \in \Sigma^+$ is a triple $\langle \ell, r, |\alpha| \rangle$ with $\ell, r \in [1, n]$ and $\ell < r$, where the following 3 properties hold.

1. The interval $\hat{A}[\ell..r)$ contains exactly the elements of $A[\ell..r)$ in increasing (by text position) order. We write $i \in \langle \ell, r, |\alpha| \rangle$ to denote $\exists x \in [\ell, r) : \hat{A}[x] = i$.
2. All the suffixes share prefix α , i.e. $\forall i \in \langle \ell, r, |\alpha| \rangle : S_i = \alpha S_{i+|\alpha|}$.
3. The context α is a Lyndon word.

The groups G_1, \dots, G_m form a partition of \hat{A} as follows. Let $G_i = \langle \ell_i, r_i, |\alpha_i| \rangle$ be the i -th group, then it holds $\ell_1 = 1$ and $r_m = n + 1$. For $i \in [2, m]$ it holds $\ell_i = r_{i-1}$. We write $G_i < G_j \iff i < j$ to denote that any suffix in G_i is lexicographically smaller than any suffix in G_j , which also means that the context of G_i is lexicographically not larger than the context of G_j . A group $\langle \ell, r, |\alpha| \rangle$ is called *final* if $\forall i \in \langle \ell, r, |\alpha| \rangle : w_\lambda(i) = \alpha$.

As mentioned earlier, the initial suffix grouping partitions the suffixes by their first symbol, which can be implemented as follows. We stably sort the array $\hat{A} = [1, 2, \dots, n]$ in increasing order, using key $S[i]$ for entry i . After that, we determine the group borders with a simple scan over the sorted suffixes. We store the groups on a stack Q , where the bottommost element is the leftmost group, and the topmost element is the rightmost group.

3.1 Phase 1 with Integer Sorting

The goal of Phase 1 is to sort the group contexts lexicographically. To this end, we refine the groups by splitting them into subgroups with possibly longer contexts. The general idea is as follows. For any index i in a group $G_k = \langle \ell, r, |\alpha| \rangle$, let $j = i + |\alpha|$ be the position right after the context. If $S_i \succ S_j$ then $w_\lambda(i) = \alpha$ (Lemma 5), and we place i into a *final* subgroup with unchanged context α . If however $S_i < S_j$, then $\alpha w_\lambda(j)$ is a Lyndon word (Lemma 5), and we place i into a subgroup with context $\alpha w_\lambda(j)$. We repeatedly refine the subgroups in the same way, until all groups are final. At the point in time at which we refine the group $G_k = \langle \ell, r, |\alpha| \rangle$, the data structures used by our Phase 1 algorithm are the following (see Figure 1b for an example):

- A stack Q contains groups G_1, \dots, G_k that form a partition of $\hat{A}[1..r)$. The groups are stored in increasing lexicographical order (the bottommost group G_1 is lex. smallest, the topmost group G_k is lex. largest).
- A stack P contains final groups F_1, \dots, F_h that form a partition of $\hat{A}[r..n]$. The groups are stored in decreasing lexicographical order (the bottommost group F_1 is lex. largest, the topmost group F_h is lex. smallest).
- Together with the array \hat{A} , the groups $G_1, \dots, G_k, F_h, \dots, F_1$ are a suffix grouping according to Definition 6.
- A length- n array **fgrp** maps suffixes to their final groups. If $\exists x \in [1, h] : i \in F_x$, then **fgrp** $[i] = x$. Otherwise, **fgrp** $[i] = \infty$. Note that **fgrp** inherently encodes information about the lexicographical order of suffixes due to $\forall i, j \in [1, n] : \text{fgrp}[i] < \text{fgrp}[j] \implies S_i \succ S_j$.

Now we describe Phase 1 in detail. The description is accompanied by pseudocode in Algorithm 1. The algorithm takes the array \hat{A} and the stack Q from the initialization as input. The stack P is initially empty, and all entries of **fgrp** are set to ∞ (lines 1–2). During the execution of the algorithm, we may mark the groups on the stack Q as *ready* (indicating that the group can easily be refined) or *final* (indicating that no further refinement is necessary). Initially, all groups are unmarked. The refinement is performed in a simple loop. While the stack Q is not empty, we pop the topmost group $G_k = \langle \ell, r, |\alpha| \rangle$ and process it (lines 3–4). Depending on the marking of the group, there are three possible cases:

■ **Algorithm 1** Phase 1 with integer sorting.

Input: Initial suffix grouping represented by array \hat{A} and stack Q (all groups unmarked).

Output: Final suffix grouping represented by array \hat{A} and stack P .

```

1:  $P \leftarrow$  empty stack
2: for  $i \in [1, n]$  do  $\text{fgrp}[i] \leftarrow \infty$ 
3: while  $Q$  is not empty do
4:    $\langle \ell, r, |\alpha| \rangle \leftarrow Q.\text{pop}()$ 
5:   if  $\langle \ell, r, |\alpha| \rangle$  is marked final  $\vee \ell = 1$  then
6:      $P.\text{push}(\langle \ell, r, |\alpha| \rangle)$ 
7:     for  $i \in [\ell, r]$  do
8:        $\text{fgrp}[\hat{A}[i]] \leftarrow |P|$ 
9:   else if  $\langle \ell, r, |\alpha| \rangle$  is marked ready then
10:    Stably sort  $\hat{A}[\ell..r]$  in decreasing order,
    using key  $\text{fgrp}[\hat{A}[i] + |\alpha|]$  for entry  $\hat{A}[i]$ .
11:     $\ell' \leftarrow \ell$ 
12:    for  $i \in (\ell, r)$  in increasing order do
13:      if  $\text{fgrp}[\hat{A}[i-1] + |\alpha|] \neq \text{fgrp}[\hat{A}[i] + |\alpha|]$  then
14:        Let  $\beta$  be the context of  $F_{\text{fgrp}[\hat{A}[i-1] + |\alpha|]}$ .
15:         $Q.\text{push}(\langle \ell', i, |\alpha\beta| \rangle)$ 
16:         $\ell' \leftarrow i$ 
17:    Let  $\beta$  be the context of  $F_{\text{fgrp}[\hat{A}[r-1] + |\alpha|]}$ .
18:     $Q.\text{push}(\langle \ell', r, |\alpha\beta| \rangle)$ 
19:   else
20:     for  $i \in [\ell, r]$  in decreasing order do
21:       if  $i < r - 1 \wedge \hat{A}[i+1] = \hat{A}[i] + |\alpha|$  then
22:         if  $\text{sg}(\hat{A}[i+1]) = \infty$  then  $\text{sg}(\hat{A}[i]) \leftarrow \infty$ 
23:         else  $\text{sg}(\hat{A}[i]) \leftarrow \text{sg}(\hat{A}[i+1]) + 1$ 
24:       else
25:         if  $\text{fgrp}[\hat{A}[i] + |\alpha|] = \infty$  then  $\text{sg}(\hat{A}[i]) \leftarrow \infty$ 
26:         else  $\text{sg}(\hat{A}[i]) \leftarrow 1$ 
27:     Stably sort  $\hat{A}[\ell..r]$  in decreasing order,
    using key  $\text{sg}(\hat{A}[i])$  for entry  $\hat{A}[i]$ .
28:      $\ell' \leftarrow \ell$ 
29:     while  $\text{sg}(\hat{A}[\ell']) = \infty$  do  $\ell' \leftarrow \ell' + 1$ 
30:     if  $\ell' > \ell$  then  $Q.\text{push}(\langle \ell, \ell', |\alpha| \rangle)$  marked final
31:     for  $i \in (\ell', r)$  in increasing order do
32:       if  $\text{sg}(\hat{A}[i-1]) \neq \text{sg}(\hat{A}[i])$  then
33:          $Q.\text{push}(\langle \ell', i, |\alpha| \rangle)$  marked ready
34:          $\ell' \leftarrow i$ 
35:      $Q.\text{push}(\langle \ell', r, |\alpha| \rangle)$  marked ready

```

final
group

ready
group

unmarked
group

The group is marked final. or it is the group containing only the lexicographically smallest suffix $S_n = \$$ (which gets processed last). During an earlier processing step, we have already ensured that $\forall i \in G_k : w_\lambda(i) = \alpha$. We simply push the group onto the final stack P , which now contains $|P| = h + 1$ groups. We update the array **fgrp** accordingly by assigning $\text{fgrp}[i] \leftarrow h + 1$ for each index $i \in G_k$ (lines 5–8).

The group is marked ready. During an earlier processing step, we have already ensured that for each index $i \in G_k$, the index $i + |\alpha|$ is contained in a group that is lexicographically larger than G_k . Since all the lexicographically larger groups are final, it holds $\text{fgrp}[i + |\alpha|] \neq \infty$. Particularly, for any two indices $i, j \in G_k$ it holds $\text{fgrp}[i + |\alpha|] > \text{fgrp}[j + |\alpha|] \implies S_i \prec S_j$ and $\text{fgrp}[i + |\alpha|] < \text{fgrp}[j + |\alpha|] \implies S_i \succ S_j$. We sort the interval $\hat{A}[\ell..r]$ in *decreasing* order, using key $\text{fgrp}[i + |\alpha|]$ for index i (line 10). Indices that share the same key x form a subgroup H_x (we again determine the group borders by scanning; lines 12–13). Let β be the context of the final group F_x , then $\alpha\beta$ becomes the context of subgroup H_x (lines 14–15 and 17–18). We push the subgroups onto the stack Q in lexicographically increasing order. All subgroups are unmarked.

The group is unmarked. We have already seen that ready and final groups are relatively easy to process. In this processing step, we split an unmarked group into at most one final subgroup H_∞ , and possibly multiple ready subgroups H_1, H_2, \dots, H_m (where m is unknown in advance). The lexicographical order of subgroups is $H_\infty \prec H_m \prec \dots \prec H_1$. All subgroups have unchanged context α ; the context extension only takes place when processing the ready subgroups. The subgroup H_∞ will contain exactly the indices $i \in G_k$ with $w_\lambda(i) = \alpha$. Lemma 4 implies that these suffixes are lexicographically smallest amongst the suffixes in G_k . Consider any index $i \in G_k$, and let x be the smallest positive integer such that $i' = i + x \cdot |\alpha| \notin G_k$. It is easy to see that $S_i = \alpha^x S_{i'}$.

- If i' is in one of the lexicographically smaller groups G_1, \dots, G_{k-1} , then $S_i \succ S_{i'}$ and simple properties of the lexicographical order imply $S_i \succ S_{i+|\alpha|} \succ S_{i+2|\alpha|} \succ \dots \succ S_{i+x|\alpha|}$. It follows from Lemma 3 that $w_\lambda(i) = \alpha$, and we place i into subgroup H_∞ . Note that if $x > 1$ and $i \in H_\infty$, then $i + |\alpha| \in H_\infty$.
- If i' is in one of the lexicographically larger groups F_1, \dots, F_h , then $S_i \prec S_{i'}$ and simple properties of the lexicographical order imply $S_i \prec S_{i+|\alpha|} \prec S_{i+2|\alpha|} \prec \dots \prec S_{i+x|\alpha|}$. It follows from Lemma 3 that $\lambda[i] > |\alpha|$, and we place i into subgroup H_x . Note that if $x > 1$ and $i \in H_x$, then $i + |\alpha| \in H_{x-1}$. Thus H_x can be marked ready.

Now we show that in fact $H_m \prec H_{m-1} \prec \dots \prec H_1$. Assume that we place two indices i and j into subgroups H_x and H_y respectively. We have to show that $(x > y \implies S_i \prec S_j)$ and $(x < y \implies S_i \succ S_j)$. If $x > y$, then $S_i = \alpha^y S_{i+y \cdot |\alpha|}$ and $S_j = \alpha^y S_{j+y \cdot |\alpha|}$. Because of $x > y$ it holds $i + y \cdot |\alpha| \in G_k$, while $j + y \cdot |\alpha|$ is in one of the lexicographically larger groups F_1, \dots, F_h . Thus it holds $S_{i+y \cdot |\alpha|} \prec S_{j+y \cdot |\alpha|}$ and therefore also $S_i \prec S_j$. The proof of $x < y \implies S_i \succ S_j$ works analogously.

As seen above, if both i and $i + |\alpha|$ are in G_k , then it holds $i + |\alpha| \in H_\infty \implies i \in H_\infty$ and $i + |\alpha| \in H_{x-1} \implies i \in H_x$. In such cases, we can easily compute i 's subgroup from $(i + |\alpha|)$'s subgroup. We only need an efficient way to check whether $i + |\alpha| \in G_k$ actually holds. Conveniently, $i + |\alpha| \in G_k$ if and only if i and $i + |\alpha|$ are *neighboring* entries in $\hat{A}[\ell..r]$. This is due to the fact that there cannot be a suffix $S_j = \alpha S_{j+|\alpha|}$ with $j \in (i, i + |\alpha|)$ (otherwise α would have a proper prefix that is also a proper suffix, which contradicts the definition of Lyndon words).

Lines 19–35 of Algorithm 1 describe our strategy for unmarked groups in technical detail. First, we compute a key $sg(\hat{A}[i])$ for each $i \in [\ell, r]$ in decreasing order, indicating that we place index $\hat{A}[i]$ into subgroup $H_{sg(\hat{A}[i])}$. If for some index $\hat{A}[i]$ it holds $\hat{A}[i] + |\alpha| \in G_k$,

i.e. if $\hat{A}[i]$ and $\hat{A}[i] + |\alpha|$ are neighbors in $\hat{A}[\ell..r]$, then we compute $\hat{A}[i]$'s subgroup from $(\hat{A}[i] + |\alpha|)$'s subgroup as described above (lines 21–23). Otherwise, we inspect $\text{fgrp}[\hat{A}[i] + |\alpha|]$ to decide whether we place $\hat{A}[i]$ into subgroup H_∞ or subgroup H_1 (lines 24–26). Finally, we rearrange $\hat{A}[\ell..r]$ according to the new subgroups (line 27). We push the subgroups onto stack Q in increasing lexicographical order (once again computing the group borders with a simple scan). If H_∞ exists, then we mark it final (lines 29–30). All other groups are marked ready (lines 31–35).

3.2 Phase 2 with Integer Sorting

In the second phase, we lexicographically sort each final group, and simultaneously compute the inverse suffix array (see Algorithm 2). We proceed similarly to the first phase, using the stack P and the array \hat{A} as input. First, we pop the topmost group of P (which is the lexicographically smallest group containing only the special suffix $S_n = \$$), and assign $A^{-1}[1] = n$ (line 1). Then we sort the remaining groups in a simple loop. While P is not empty, we pop the topmost group $F_h = \langle \ell, r, |\alpha| \rangle$ of the stack (lines 2–3) and lexicographically sort it. At this point in time, we have already sorted the suffixes of all groups that are lexicographically smaller than F_h because P contains the groups in lexicographical order (the topmost group is the lexicographically smallest unsorted group). Therefore, when processing F_h we have $\forall i \in [1, \ell) : A^{-1}[A[i]] = i$.

Since the group is final, it holds $S_{\hat{A}[i]} \succ S_{\hat{A}[i] + |\alpha|}$ for each $i \in [\ell, r)$. Thus $\hat{A}[i] + |\alpha|$ either was in one of the lexicographically smaller groups that we have already sorted, or it holds $\hat{A}[i] + |\alpha| \in F_h$. Ideally, we would simply sort $\hat{A}[\ell..r]$ using key $A^{-1}[\hat{A}[i] + |\alpha|]$ for entry $\hat{A}[i]$. However, if $\hat{A}[i] + |\alpha| \in F_h$, then we have not computed $A^{-1}[\hat{A}[i] + |\alpha|]$ yet. We solve this problem by first rearranging F_h into subgroups $H_1 \prec H_2 \prec \dots \prec H_m$, where subgroup H_x contains the suffixes that have prefix α^x , but not prefix α^{x+1} (the correctness of the lexicographical order of these subgroups can be shown similarly to the order of subgroups for unmarked groups in Phase 1). We assign the indices to the subgroups in decreasing order (line 4). If $\hat{A}[i] + |\alpha| \notin F_h$ (as before, this is the case if and only if $\hat{A}[i]$ and $\hat{A}[i] + |\alpha|$ are not neighbors in $\hat{A}[\ell..r]$), then we place $\hat{A}[i]$ into subgroup H_1 (line 6). Otherwise, we have already placed $\hat{A}[i] + |\alpha|$ into some subgroup H_{x-1} , and we place i into subgroup H_x (line 5). After we have rearranged the indices according to the new grouping (line 7), we finally sort each subgroup using key $A^{-1}[\hat{A}[i] + |\alpha|]$ for entry $\hat{A}[i]$ (lines 8–15). Whenever we sort a group, we also update the inverse suffix array. Since no two indices $\hat{A}[i]$ and $\hat{A}[i] + |\alpha|$ are in the same subgroup, and due to the lexicographical order of subgroups, the required keys are always available once they are needed.

4 Implementation Details

Our C++17 implementation of the algorithm is publicly available on GitHub². In general, it closely follows the description from Section 3. In this section, we discuss the choice of integer sorters as well as other practical optimizations, including the parallelization of the algorithm.

Sequential Implementation. The main computational effort of the algorithm lies in integer sorting, as well as in the computation of the keys prior to sorting. Finding the group borders after sorting only requires simple sequential scans that are cache efficient and very fast in

² <https://github.com/jonas-ellert/gsaca-double-sort>

■ **Algorithm 2** Phase 2 with integer sorting.

Input: Final suffix grouping represented by array \hat{A} and stack P .

Output: Suffix array \hat{A} and inverse suffix array A^{-1} .

```

1:  $P.pop()$ ;  $A^{-1}[1] \leftarrow n$ ;
2: while  $P$  is not empty do
3:    $\langle \ell, r, |\alpha| \rangle \leftarrow P.pop()$ 
4:   for  $i \in [\ell, r]$  in decreasing order do
5:     if  $i < r - 1 \wedge \hat{A}[i + 1] = \hat{A}[i] + |\alpha|$  then  $sg(\hat{A}[i]) \leftarrow sg(\hat{A}[i + 1]) + 1$ 
6:     else  $sg(\hat{A}[i]) \leftarrow 1$ 
7:   Stably sort  $\hat{A}[\ell..r]$  in increasing order, using key  $sg(\hat{A}[i])$  for entry  $\hat{A}[i]$ .
8:    $\ell' \leftarrow \ell$ 
9:   for  $i \in (\ell, r)$  in increasing order do
10:    if  $sg(\hat{A}[i - 1]) \neq sg(\hat{A}[i])$  then
11:      Sort  $\hat{A}[\ell'..i]$  in increasing order, using key  $A^{-1}[\hat{A}[i] + |\alpha|]$  for entry  $\hat{A}[i]$ .
12:      for  $j \in [\ell'..i]$  do  $A^{-1}[\hat{A}[j]] \leftarrow i$ 
13:       $\ell' \leftarrow i$ 
14:   Sort  $\hat{A}[\ell'..r]$  in increasing order, using key  $A^{-1}[\hat{A}[i] + |\alpha|]$  for entry  $\hat{A}[i]$ .
15:   for  $j \in [\ell'..r]$  do  $A^{-1}[\hat{A}[j]] \leftarrow i$ 

```

practice. We use two different sorting algorithms. Whenever the keys are from a small domain, we use a simple counting sort. This applies to the initialization (in practice we assume the byte alphabet $[0, 255]$), the processing of unmarked groups in the first phase, and the computation of subgroups in the second phase (Algorithm 1, line 27 and Algorithm 2, line 7). In both of the latter cases, we simultaneously compute the keys and count their frequencies. When refining ready groups in the first phase (Algorithm 1, line 10), and when performing the final sorting in the second phase (Algorithm 2, lines 11 and 14), we use MSD radix sort. We focus on speed and thus do not use an in-place variant of the sorter. Therefore, apart from the space needed for the suffix array, one auxiliary array (for $fgrp$ and A^{-1}), and the stacks, we need additional space linear in the size of the largest group that we encounter.

We provide three versions DS1, DS3, and DSH of our sequential implementation that differ only in the initialization (where DS stands for *double sort* because we reduced both phases to integer sorting). The first version DS1 corresponds to the description in Section 3, i.e. we sort the suffixes by their first symbol. The second version DS3 directly sorts the suffixes by their first three symbols, resulting in smaller groups after the initialization (potentially resulting in a smaller memory footprint). Finally, the version DSH directly sorts the suffixes using key $w_\lambda(i)$ for suffix S_i . However, if $\lambda[i] > 8$ then we use key $w_\lambda(i)[1..8]$. Thus we immediately place all suffixes with $\lambda[i] < 8$ into their correct final groups, skipping many refinement steps and therefore accelerating Phase 1. The initialization of DSH consists of three steps.

Extract: We use a modification of Duval’s algorithm [5] to compute for each $i \in [1, n]$ the key $x(i) = w_\lambda(i)$, if $\lambda[i] \leq 8$, or $x(i) = w_\lambda(i)[1..8]$ otherwise. We then use a hash table³ to check whether this is the first time we have seen key $x(i)$. If it is, then we add the

³ We use an implementation of Robin Hood hashing by Martin Ankerl, see <https://github.com/martinus/robin-hood-hashing>

tuple $\langle x(i), i \rangle$ to the table. Otherwise, let j be the index where we first discovered key $x(i)$. We store $\hat{A}[i] = j$, indicating that we will place i into the same group as j .

Sort: We use comparison sorting to sort the tuples $\langle x(i), i \rangle$ by their first component.

Rearrange: We group the indices according to the sorted keys, and push the groups onto the stack Q . If a group has context α with $|\alpha| < 8$ then we mark it final.

Parallel Implementation. For our parallel algorithms we implemented the same ideas as described in Section 3 using OpenMP for parallelization. We parallelized the counting sort in the initialization as well as the processing of each suffix group in Phases 1 and 2. Because processing a single suffix group consists mainly of integer sorting and sequential scans we can parallelize them well in practice. We replace the sequential scans with parallel prefix sums. For integer sorting we use `ips4o` [1]. In case we have to sort a range $\hat{A}[\ell..r)$ of elements stably we sort $\hat{A}[\ell..r)$ by their keys, breaking ties by sorting elements with equal keys by their entry $\hat{A}[i]$. Since for small suffix groups the overhead to process the group in parallel is too high compared to the sequential implementation we use a threshold. If the size of a suffix group is at most 1024, we switch to the sequential implementation (we performed a preliminary evaluation to determine a value that performs well in practice).

Unfortunately, we cannot achieve a perfect speedup. We simply cannot process multiple suffix groups in parallel because we heavily rely on processing them in decreasing lexicographical order. For that reason the running time of our parallelization is still linear with respect to the number of processed suffix groups (which is n in the worst case).

Similarly to our sequential implementation, we provide two different parallel versions PDS1 and PDS2. In PDS1 we sort the suffixes in the initialization by their first symbol using a parallel counting sort. In PDS2 we sort the suffixes by their first two symbols using `ips4o`.

5 Experimental Evaluation

We evaluated our algorithms on a number of real and artificial texts taken from the Pizza & Chili text corpus⁴. In Table 1 we give an overview of the used texts. We divide the texts into three different categories. Real texts (PC-Real) include `english`, `dna`, `sources`, `proteins` and `dblp.xml`. They are good examples of texts that occur in real-world applications. Real repetitive texts (PC-Rep-Real) include `cere`, `einstein.en.txt`, `kernel` and `para`. These texts might still occur in real-world applications, but they are rather repetitive and thus highly compressible. And lastly, artificial repetitive texts (PC-Rep-Art) include `fib41`, `rs.13` and `tm29`. These texts were created artificially with the goal of repetitiveness in mind. All the aforementioned texts are sufficiently short to compute the 32-bit suffix array. For our weak scaling experiments we use a collection of larger texts (**Large**) that are summarized at the bottom of Table 1. From each text we use a prefix of up to 16 GiB for our experiments, which means that 32 bits are not sufficient to address the suffixes. Thus, we compute the 64-bit suffix array instead.

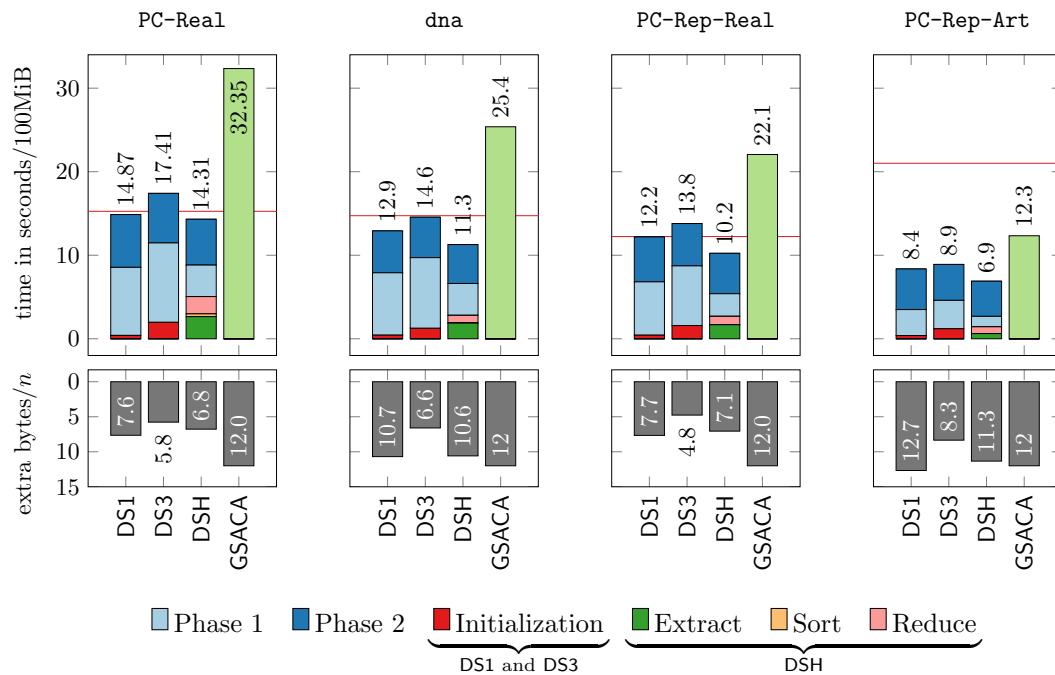
5.1 Experimental Setup and Results

We conducted our experiments on a Linux machine with an AMD EPYC 7452 processor (32 cores, 2.35 GHz, L1 32K, L2 512K, L3 16M) and 1 TB of RAM. The code was compiled using GCC 7.5.0. We repeated all of our experiments five times and use the median as the

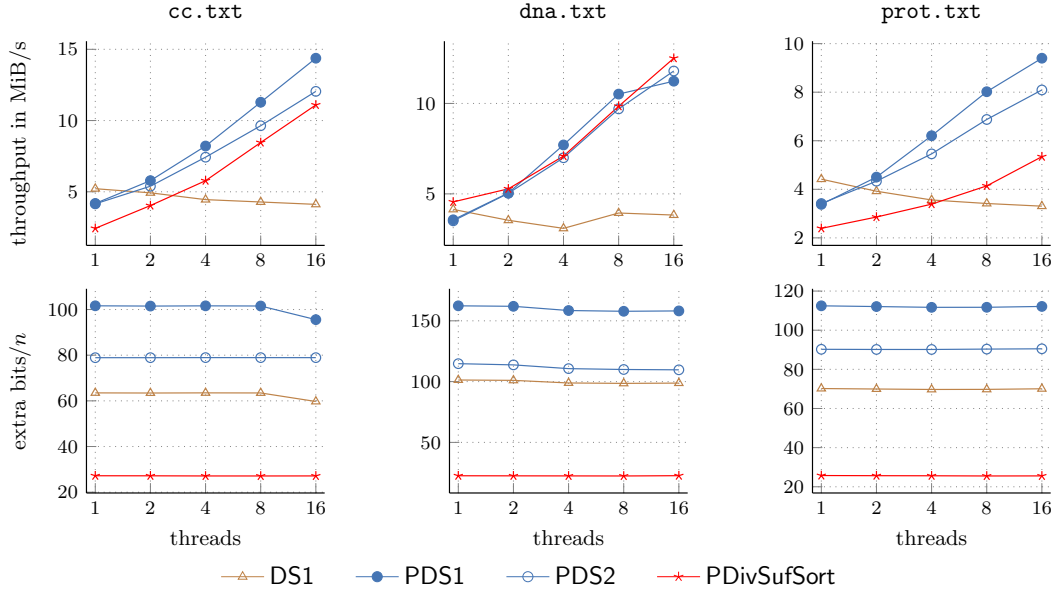
⁴ <http://pizzachili.dcc.uchile.cl/>

■ **Table 1** Texts from the Pizza & Chili text corpus and large texts. Apart from the text size n and the alphabet size σ , we provide the number of suffix groups (SGs) and the average size of the suffix groups that we process in Phase 1.

Category	Text	n (MiB)	σ	Count of SGs	Avg. SG Size
PC-Real	english	1,024	237	102,640,785	40.72
	dna	386	16	44,436,473	34.49
	sources	202	230	27,961,111	28.92
	proteins	1,024	27	144,323,711	28.62
	dblp.xml	283	97	16,091,809	71.9
PC-Rep-Real	cere	440	5	10,585,469	174.06
	einstein.en.txt	446	139	409,384	4,568.34
	kernel	247	160	4,027,324	256.15
	para	410	5	12,241,149	139.98
PC-Rep-Art	fib41	256	2	98	10,935,276.92
	rs.13	207	2	197	4,400,958.45
	tm29	256	2	194	5,534,751.23
Large	cc.txt	32,768	243	1,528,252,068	88.53
	dna.txt	32,768	4	2,955,252,973	44.56
	prot.txt	32,768	26	4,308,219,247	30.47



■ **Figure 2** Running time and additional memory usage of the sequential algorithms averaged for each text category, and additionally for the text **dna**. The red line marks the running time of DivSufSort. We do not show the memory usage of DivSufSort because it requires only a very small constant amount of additional working space. (Best viewed in color.)



■ **Figure 3** Throughput and additional memory usage for each parallel algorithm on large texts. The text size scales with the number of used threads (1GiB per thread). (Best viewed in color.)

final result. For the sequential experiments we compare our algorithms DS1, DS3, and DSH with Baier’s original implementation of GSACA [4] and the currently fastest suffix sorter DivSufSort [6]. For the weak scaling experiments we compare our parallel algorithms PDS1 and PDS2 with Labeit’s parallel implementation PDivSufSort of DivSufSort [12], and with our sequential DS1 as a baseline to see how well our parallelization scales.

Sequential Results. For each of the categories PC-Real, PC-Rep-Real and PC-Rep-Art, we averaged the running time and additional memory usage of the texts in each category. Before computing the average, we normalized the running time for each algorithm to show how long the algorithms take for 100 MiB of the input text. The additional memory is normalized as well to show the additional memory usage for each byte in the input text. The results for each category can be seen in Figure 2. Additionally, we provide a separate plot for *dna*, which is one of the most relevant text types in practice.

All of our sequential algorithms are significantly faster than Baier’s original implementation. Our fastest sequential algorithm DSH is around twice as fast as Baier’s algorithm, and is in each category even faster than DivSufSort (over 6% faster on PC-Real, over 19% faster on PC-Rep-Real, and over three times as fast on PC-Rep-Art). On PC-Real and PC-Rep-Real, the additional memory usage of our algorithms is much lower than Baier’s algorithm. Of our new algorithms, DS3 has the lowest memory usage, but the running time is not as good as DS1 and DSH. None of our algorithms comes close to the memory usage of DivSufSort, which only uses a very small amount of constant additional memory.

Weak Scaling Results. Figure 3 shows the results of our weak scaling experiments. We calculate for each text the throughput (text size divided by running time) in MiB/s and the additional memory in bits/n for up to 16 threads. The input size grows proportionally to the number of used threads, such that the input size is p GiB when using p threads. We do not include results for more than 16 threads because otherwise the memory usage exceeds the maximum amount of 500 GB memory that we can address on a single NUMA node, and in some cases even the total memory of 1 TB (which is a limitation of our algorithms).

On `cc.txt` and `prot.txt`, our parallel variants are significantly faster than PDivSufSort for all input sizes. On `dna.txt`, the performance of all algorithms is similar. The memory usage of PDS1 is up to five times as large as PDivSufSort and the memory usage of PDS2 is up to four times as large as PDivSufSort on all input texts. The memory usage of DS1 is on all input texts lower than PDS1. This is due to the fact that the sequential algorithm uses 40-bit types for the additional arrays `fgrp` and A^{-1} . The parallel algorithms however use 64-bit types because the running time gets slower when using 40-bit types in parallel.


References

- 1 M. Axtmann, S. Witt, D. Ferizovic, and P. Sanders. In-place parallel super scalar samplesort (IPS⁴o). In *25th European Symposium on Algorithms (ESA)*, pages 9:1–9:14, 2017. URL: <https://arxiv.org/abs/1705.02257>.
- 2 Johannes Bahne, Nico Bertram, Marvin Böcker, Jonas Bode, Johannes Fischer, Hermann Foot, Florian Grieskamp, Florian Kurpicz, Marvin Löbel, Oliver Magiera, Rosa Pink, David Piper, and Christopher Poeplau. SACABench: Benchmarking suffix array construction. In *Proceedings of the 26th International Symposium on String Processing and Information Retrieval (SPIRE 2019)*, pages 407–416, Segovia, Spain, 2019. doi:10.1007/978-3-030-32686-9_29.
- 3 Uwe Baier. Linear-time suffix sorting - A new approach for suffix array construction. Master's thesis, Ulm University, 2015.
- 4 Uwe Baier. Linear-time suffix sorting - A new approach for suffix array construction. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, Tel Aviv, Israel, 2016. doi:10.4230/LIPIcs.CPM.2016.23.
- 5 Jean Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.
- 6 Johannes Fischer and Florian Kurpicz. Dismantling DivSufSort. In *Proceedings of the 21st Prague Stringology Conference (PSC 2019)*, pages 62–76, Prague, Czech Republic, August 2017. URL: <http://www.stringology.org/event/2017/p07.html>.
- 7 Frantisek Franek, A. S. M. Sohiddul Islam, Mohammad Sohel Rahman, and William F. Smyth. Algorithms to compute the Lyndon array. In *Proceedings of the 20th Prague Stringology Conference (PSC 2016)*, pages 172–184, Prague, Czech Republic, August 2016. URL: <http://www.stringology.org/event/2016/p15.html>.
- 8 Frantisek Franek, Asma Paracha, and William F. Smyth. The linear equivalence of the suffix array and the partially sorted Lyndon array. In *Proceedings of the 21st Prague Stringology Conference (PSC 2017)*, pages 77–84, Prague, Czech Republic, August 2017. URL: <http://www.stringology.org/event/2017/p08.html>.
- 9 Keisuke Goto. Optimal time and space construction of suffix arrays and LCP arrays for integer alphabets. In *Proceedings of the 23rd Prague Stringology Conference (PSC 2019)*, pages 111–125, Prague, Czech Republic, 2019. URL: <http://www.stringology.org/event/2019/p11.html>.
- 10 Torben Hagerup. Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*, pages 366–398, Paris, France, 1998. doi:10.1007/BFb0028575.
- 11 Florian Kurpicz. *Parallel Text Index Construction*. PhD thesis, Technical University of Dortmund, 2020.
- 12 Julian Labeit, Julian Shun, and Guy E. Blelloch. Parallel lightweight wavelet tree, suffix array and FM-index construction. *Journal of Discrete Algorithms*, 43:2–17, 2017. doi:10.1016/j.jda.2017.04.001.
- 13 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the 1st Annual Symposium on Discrete Algorithms (SODA 1990)*, pages 319–327, San Francisco, California, USA, 1990. URL: <https://dl.acm.org/doi/10.5555/320176.320218>.
- 14 Simon J. Puglisi, William F. Smyth, and Andrew Turpin. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2):4, 2007. doi:10.1145/1242471.1242472.

Online Euclidean Spanners

Sujoy Bhore 

Indian Institute of Science Education and Research, Bhopal, India

Csaba D. Tóth 

California State University Northridge, Los Angeles, CA, USA

Tufts University, Medford, MA, USA

Abstract

In this paper, we study the online Euclidean spanners problem for points in \mathbb{R}^d . Given a set S of n points in \mathbb{R}^d , a t -spanner on S is a subgraph of the underlying complete graph $G = (S, \binom{S}{2})$, that preserves the pairwise Euclidean distances between points in S to within a factor of t , that is the *stretch factor*. Suppose we are given a sequence of n points (s_1, s_2, \dots, s_n) in \mathbb{R}^d , where point s_i is presented in step i for $i = 1, \dots, n$. The objective of an online algorithm is to maintain a geometric t -spanner on $S_i = \{s_1, \dots, s_i\}$ for each step i . The algorithm is allowed to *add* new edges to the spanner when a new point is presented, but cannot *remove* any edge from the spanner. The performance of an online algorithm is measured by its competitive ratio, which is the supremum, over all sequences of points, of the ratio between the weight of the spanner constructed by the algorithm and the weight of an optimum spanner. Here the weight of a spanner is the sum of all edge weights.

First, we establish a lower bound of $\Omega(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$ for the competitive ratio of any online $(1 + \varepsilon)$ -spanner algorithm, for a sequence of n points in 1-dimension. We show that this bound is tight, and there is an online algorithm that can maintain a $(1 + \varepsilon)$ -spanner with competitive ratio $O(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$. Next, we design online algorithms for sequences of points in \mathbb{R}^d , for any constant $d \geq 2$, under the L_2 norm. We show that previously known incremental algorithms achieve a competitive ratio $O(\varepsilon^{-(d+1)} \log n)$. However, if the algorithm is allowed to use additional points (Steiner points), then it is possible to substantially improve the competitive ratio in terms of ε . We describe an online Steiner $(1 + \varepsilon)$ -spanner algorithm with competitive ratio $O(\varepsilon^{(1-d)/2} \log n)$. As a counterpart, we show that the dependence on n cannot be eliminated in dimensions $d \geq 2$. In particular, we prove that any online spanner algorithm for a sequence of n points in \mathbb{R}^d under the L_2 norm has competitive ratio $\Omega(f(n))$, where $\lim_{n \rightarrow \infty} f(n) = \infty$. Finally, we provide improved lower bounds under the L_1 norm: $\Omega(\varepsilon^{-2} / \log \varepsilon^{-1})$ in the plane and $\Omega(\varepsilon^{-d})$ in \mathbb{R}^d for $d \geq 3$.

2012 ACM Subject Classification Mathematics of computing \rightarrow Approximation algorithms; Mathematics of computing \rightarrow Paths and connectivity problems; Theory of computation \rightarrow Computational geometry

Keywords and phrases Geometric spanner, $(1 + \varepsilon)$ -spanner, minimum weight, online algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.16

Related Version Full Version: [arXiv:2107.00684](https://arxiv.org/abs/2107.00684)

Funding Csaba D. Tóth: Research was partially supported by the NSF award DMS-1800734.

1 Introduction

We study the online Euclidean spanners problem for a set of points in \mathbb{R}^d . Let S be a set of n points in \mathbb{R}^d . A t -spanner for a finite set S of points in \mathbb{R}^d is a subgraph of the underlying complete graph $G = (S, \binom{S}{2})$, that preserves the pairwise Euclidean distances between points in S to within a factor of t , that is the *stretch factor*. The edge weights of G are the Euclidean distances between the vertices. Chew [22, 23] initiated the study of Euclidean spanners in 1986, and showed that for a set of n points in \mathbb{R}^2 , there exists a spanner with $O(n)$ edges and constant stretch factor. Since then a large body of research has been



© Sujoy Bhore and Csaba D. Tóth;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 16; pp. 16:1–16:19

Leibniz International Proceedings in Informatics



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

devoted to Euclidean spanners due to its vast applications across domains, such as, topology control in wireless networks [50], efficient regression in metric spaces [31], approximate distance oracles [36], and many others. Moreover, Rao and Smith [48] showed the relevance of Euclidean spanners in the context of other fundamental geometric NP-hard problems, e.g., Euclidean traveling salesman problem and Euclidean minimum Steiner tree problem. Many different spanner construction approaches have been developed for Euclidean spanners over the years, that each found further applications in geometric optimization, such as spanners based on well-separated pair decomposition (WSPD) [17, 35], skip-lists [4], path-greedy and gap-greedy approaches [3, 5], locality-sensitive orderings [21], and more. We refer to the book by Narasimhan and Smid [47] and the survey of Bose and Smid [16] for a summary of results and techniques on Euclidean spanners up to 2013.

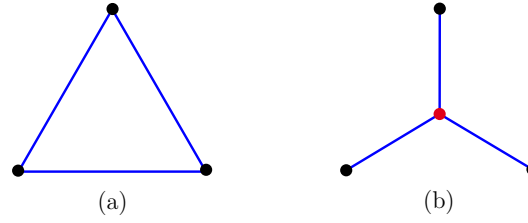
Online Spanners. We are given a sequence of n points (s_1, s_2, \dots, s_n) , where the points are presented one-by-one, i.e., point s_i is revealed at the step i , and $S_i = \{s_1, \dots, s_i\}$ for $i = 1, \dots, n$. The objective of an online algorithm is to maintain a geometric t -spanner G_i for S_i for all i . Importantly, the algorithm is allowed to *add* edges to the spanner when a new point arrives, however is not allowed to *remove* any edge from the spanner.

The performance of an online algorithm ALG is measured by comparing it to the offline optimum OPT using the standard notion of competitive ratio [14, Ch. 1]. The *competitive ratio* of an online t -spanner algorithm ALG is defined as $\sup_{\sigma} \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)}$, where the supremum is taken over all input sequences σ , $\text{OPT}(\sigma)$ is the minimum weight of a t -spanner for σ , and $\text{ALG}(\sigma)$ denotes the weight of the t -spanner produced by ALG for this input.

Computing a $(1+\varepsilon)$ -spanner of minimum weight for a set S in Euclidean plane is known to be NP-hard [20]. However, there exists a plethora of constant-factor approximation algorithms for this problem in the offline model; see [3, 25, 26, 48]. Most of these algorithms approximate the parameter *lightness* (the ratio of the spanner weight to the weight of the Euclidean minimum spanning tree $\text{MST}(S)$) of Euclidean spanners, which in turn also approximates the optimum weight of the spanner. We refer to Section 1.1 for a more detailed overview of the parameter lightness.

Minimum spanning trees (MST) on n points in a metric space, which have no guarantee on the stretch factor, have been studied in the online model. It is not difficult to show that a greedy algorithm achieves a competitive ratio $\Theta(\log n)$. The online Steiner tree problem was studied by Imase and Waxman [39], who proved $\Theta(\log n)$ -competitiveness for the problem. Later, Alon and Azar [2] studied minimum Steiner trees for points in the Euclidean plane, and proved a lower bound $\Omega(\log n / \log \log n)$ for the competitive ratio. Their result was the first to analyse the impact of Steiner points on a geometric network problem in the online setting. Several algorithms were proposed over the years for the online Steiner Tree and Steiner forest problems, on graphs in both weighted and unweighted settings; see [1, 6, 10, 37, 46].

Online Steiner Spanners. An important variant of online spanners is when it is allowed to use auxiliary points (Steiner points) which are not part of input sequence of points. It turns out that Steiner points allow for substantial improvements over the bounds on the sparsity and lightness of Euclidean spanners in the offline settings; see [12, 13, 42, 43]. In the geometric setting, an online algorithm is allowed to *add* Steiner points and *subdivide* existing edges with Steiner points at each time step. (This modeling decision has twofold justification: It accurately models physical networks such as roads, canals, or power lines, and from the theoretical perspective, it is hard to tell whether an online algorithm introduced a large number of Steiner points when it created an edge/path in the first place). However, the spanner must achieve the given stretch factor only for the input point pairs.



■ **Figure 1** (a) An optimum $\frac{3}{2}$ -spanner on three points with all edges of unit length. (b) After inserting a fourth point at the center, the weight of the optimum $\frac{3}{2}$ -spanner decreases.

It is easy to see that in this model the online spanners in 1-dimension could attain optimum competitive ratio. However, it is unclear how it extends to higher dimensions as it has been observed in the offline settings that it tends to be more difficult to achieve tight bounds for Steiner spanners than their non-Steiner counterparts.

When the optimal Steiner spanner is lighter than $\text{OPT}(S_i)$ without Steiner points, the adversary may decrease $\text{OPT}(S_i)$ by adding suitable Steiner vertices to S_i ; see Fig. 1. In particular, $\text{OPT}(S_i)$ may or may not increase with i in the model without Steiner points, but $\text{OPT}(S_i)$ monotonically increases in i when Steiner points are allowed.

1.1 Related Work

Dynamic Spanners. In applications, the data (modeled as points in \mathbb{R}^d) changes over time, as new cities emerge, new wireless antennas are built, and users turn their wireless devices on or off. *Dynamic* models aim to maintain a geometric t -spanners for a dynamically changing point set S ; in a restricted *insert-only* model, the input consists of a sequence of point insertions. In the dynamic model, the objective is design algorithms and data structures that minimize the worst-case update time needed to maintain a t -spanner for S over all steps, regardless of its weight, sparsity, or lightness. Notice that dynamic algorithms are allowed to add or delete edges in each step, while online algorithms cannot delete edges. However, if a dynamic (or dynamic insert-only) algorithm always adds edges for a sequence of points insertions, it is also an online algorithm, and one can analyze its competitive ratio.

Arya et al. [4] designed a randomized incremental algorithm for n points in \mathbb{R}^d , where the points are inserted in a random order, and maintains a t -spanner of $O(n)$ size and $O(\log n)$ diameter. Their algorithm can also handle random insertions and deletions in $O(\log^d n \log \log n)$ expected amortized update time. Later, Bose et al. [15] presented an insert-only algorithm to maintain a t -spanner of $O(n)$ size and $O(\log n)$ diameter in \mathbb{R}^d . Fischer and Har-Peled [29] used dynamic compressed quadtrees to maintain a WSPD-based $(1+\varepsilon)$ -spanner for n points in \mathbb{R}^d in expected $O([\log n + \log \varepsilon^{-1}] \varepsilon^{-d} \log n)$ update time. Their algorithm works under the online model, too, however, they have not analyzed the weight of the resulting spanner. Gao et al. [30] used hierarchical clustering for dynamic spanners in \mathbb{R}^d . Their DEFSPANER algorithm is fully dynamic with $O(\log \Delta)$ update time, where Δ is the spread¹ of the set S . They maintain a $(1+\varepsilon)$ -spanner of weight $O(\varepsilon^{-(d+1)} \|MST(S)\| \log \Delta)$, and for a sequence of point insertions, DEFSPANER only adds edges. As $\text{OPT} \geq \|MST(S)\|$, DEFSPANER can serve as an online algorithm with competitive ratio $O(\varepsilon^{-(d+1)} \log \Delta)$.

¹ The *spread* of a finite set S in a metric space is the ratio of the maximum pairwise distance to the minimum pairwise distance of points in S ; and $\log \Delta \geq \Omega(\log n)$ in doubling dimensions.

Gottlieb and Roditty [32] studied dynamic spanners in more general settings. For every set of n points in a metric space of bounded doubling dimension², they constructed a $(1 + \varepsilon)$ -spanner whose maximum degree is $O(1)$ and that can be maintained under insertions and deletions in $O(\log n)$ amortized update time per operation. Later, Roditty [49] designed fully dynamic geometric t -spanners with optimal $O(\log n)$ update time for n points in \mathbb{R}^d . Very recently, Chan et al. [21] introduced *locality sensitive orderings* in \mathbb{R}^d , which has applications in several proximity problems, including spanners. They obtained a fully dynamic data structure for maintaining a $(1 + \varepsilon)$ -spanners in Euclidean space with logarithmic update time and linearly many edges. However, the spanner weight has not been analyzed for any of these constructions. Dynamic spanners have been subject to investigation in abstract graphs, as well. See [8, 9, 11] for some recent progress on dynamic graph spanners.

Lightness and **sparsity** are two natural parameters for Euclidean spanners. For a set S of points in \mathbb{R}^d , the lightness is the ratio of the spanner weight (i.e., the sum of all edge weights) to the weight of the Euclidean minimum spanning tree $MST(S)$. It is known that *greedy-spanner* ([3]) has constant lightness; see [25, 26]. Later, Rao and Smith [48] in their seminal work, showed that the greedy spanner has lightness $\varepsilon^{-O(d)}$ in \mathbb{R}^d for every constant d , and asked what is the best possible constant in the exponent. Then, the *sparsity* of a spanner on S is the ratio of its size to the size of a spanning tree. Classical results [23, 24, 40, 53] show that when the dimension $d \in \mathbb{N}$ and $\varepsilon > 0$ are constant, every set S of n points in d -space admits an $(1 + \varepsilon)$ -spanners with $O(n)$ edges and weight proportional to that of the Euclidean MST of S .

Dependence on $\varepsilon > 0$ for constant dimension d . The dependence of the lightness and sparsity on $\varepsilon > 0$ for constant $d \in \mathbb{N}$ has been studied only recently. Le and Solomon [42] constructed, for every $\varepsilon > 0$ and constant $d \in \mathbb{N}$, a set S of n points in \mathbb{R}^d for which any $(1 + \varepsilon)$ -spanner must have lightness $\Omega(\varepsilon^{-d})$ and sparsity $\Omega(\varepsilon^{-d+1})$, whenever $\varepsilon = \Omega(n^{-1/(d-1)})$. Moreover, they showed that the greedy $(1 + \varepsilon)$ -spanner in \mathbb{R}^d has lightness $O(\varepsilon^{-d} \log \varepsilon^{-1})$. In fact, Le and Solomon [42] noticed that Steiner points can substantially improve the bound on the lightness and sparsity of an $(1 + \varepsilon)$ -spanner. For minimum sparsity, they gave an upper bound of $O(\varepsilon^{(1-d)/2})$ for d -space and a lower bound of $\Omega(\varepsilon^{-1/2} / \log \varepsilon^{-1})$. For minimum lightness, they gave a lower bound of $\Omega(\varepsilon^{-1} / \log \varepsilon^{-1})$, for points in the plane ($d = 2$) [42]. More recently, Bhore and Tóth [13] established a lower bound of $\Omega(\varepsilon^{-d/2})$ for the lightness of Steiner $(1 + \varepsilon)$ -spanners in Euclidean d -space for all $d \geq 2$. Moreover, for points in the plane, they established an upper bound of $O(\varepsilon^{-1})$ [12].

1.2 Our Contributions

We present the main contributions of this paper, and sketch the key technical and conceptual ideas used for establishing these results. (Refer to the technical sections for precise definitions, complete proofs, and additional remarks.)

Points on a line. In Section 2 (Theorem 3), we establish a lower bound $\Omega(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$ for the competitive ratio of any online algorithm for a sequence of points on the real line. Moreover, we show that this bound is tight. We present an online algorithm that maintains a $(1 + \varepsilon)$ -spanner with competitive ratio $O(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$.

² A metric is said to be of a *constant doubling dimension* if a ball with radius r can be covered by at most a constant number of balls of radius $r/2$.

Our online algorithm is a 1-dimensional instantiation of hierarchical clustering, which was used by Roditty [49] for dynamical spanners in doubling metrics. When a new point s_i is “close” to a previous point s_j , we add s_i to the “cluster” of s_j , otherwise we open a new cluster. The key question is to define when s_i is “close” to a previous point. Instead of the closest points on the line, we find the shortest edge pq that contains s_i in the current spanner, and say that s_i is “close” to p (resp., q) if $\|ps_i\| \leq \frac{\varepsilon}{4}\|pq\|$ (resp., $\|qs_i\| \leq \frac{\varepsilon}{4}\|pq\|$). The algorithm (and its analysis), does not explicitly maintain “clusters,” though. It is easy to show, by induction, that ALG maintains a $(1 + \varepsilon)$ -spanner. The main contribution is a tight analysis of the competitive ratio. We partition the edges into *buckets* by weight, where bucket E_ℓ contains edges e of weight $\varepsilon^{-(\ell+1)} < \|e\| \leq \varepsilon^{-\ell}$. The edges of the spanner will form a laminar family (any edges are interior-disjoint or one contains the other); and the edge weight decay by factors of at most $(1 - \frac{\varepsilon}{4})$ along the descending paths in the containment poset. Since $(1 - \frac{\varepsilon}{4})^{4/\varepsilon} < \frac{1}{2}$, we can show that the total weight of edges in a level decreases by a factor of $\frac{1}{2}$ after every $\lceil 5/\varepsilon \rceil$ levels. Thus, the sum of edge weights in a *block* of $\lceil 5/\varepsilon \rceil$ consecutive levels is $O(\varepsilon^{-1}\text{OPT})$. This bound, applied to $O(\log_{\varepsilon^{-1}} n) = O(\log n / \log \varepsilon^{-1})$ buckets, proves the upper bound. The lower bound construction matches the upper bound for each block of levels and for each bucket.

Euclidean d -space without Steiner points. In Section 3, we study the online Euclidean spanners for a sequence of points in \mathbb{R}^d . For constant $d \geq 2$ and parameter $\varepsilon > 0$, we show that the dynamic algorithm by Fischer and Har-Peled achieves, in the online model, competitive ratio $O(\varepsilon^{-(d+1)} \log n)$ for n points in \mathbb{R}^d (Theorem 4 in Section 3.1), matching the competitive ratio of DEFSPANNER by Gao et al. [30, Lemma 3.8].

The new competitive analysis of this algorithm is instrumental for extending the algorithm and its analysis to online Steiner $(1 + \varepsilon)$ -spanners (see below). We briefly describe a key geometric insight. It is well known that for $a, b \in \mathbb{R}^d$, any ab -path of weight at most $(1 + \varepsilon)\|ab\|$ lies in an ellipsoid B_{ab} with foci a and b and great axes $(1 + \varepsilon)\|ab\|$. Summation over *disjoint* ellipsoids gives a lower bound for OPT. Unfortunately, ellipsoids B_{ab} for all pairs $ab \in S$ may heavily overlap. Recently, Bhore and Tóth [13, Lemma 3] proved that any ab -path of weight at most $(1 + \varepsilon)\|ab\|$ must contain edge of total weight at least $\frac{1}{2}\|ab\|$ that are “near-parallel” to ab (technically, they make an angle at most $\varepsilon^{1/2}$ with ab); see Fig. 4(right). By partitioning the edges of the unknown OPT spanner by *both* directions and disjoint ellipsoids, we obtain a bound of $\frac{\text{ALG}}{\text{OPT}} \leq O(\varepsilon^{-(d+1)} \log n)$.

Euclidean d -space with Steiner points. When we are allowed to use Steiner points, we can substantially improve the competitive ratio in terms of ε : We describe an algorithm with competitive ratio $O(\varepsilon^{(1-d)/2} \log n)$ (Theorem 5 in Section 3.2).

The online Steiner algorithm adds a secondary layer to the non-Steiner algorithm: For each edge ab of the non-Steiner spanner G_1 , we maintain a path of weight $(1 + \varepsilon)\|ab\|$ with Steiner points; the stretch factor of the resulting Steiner spanner G_2 is $(1 + \varepsilon)^2 < (1 + 3\varepsilon)$. The key idea is to reduce the weight to maintain *buckets* of edges of G_1 that have roughly the same direction and weight, and are nearby locations; and we construct a common Steiner network N for them. Importantly, we can construct a “backbone” of the network N when the first edge ab in a bucket arrives, and we have $\|N\| \leq O(\varepsilon^{(1-d)/2}\|ab\|)$. When subsequent edges $a'b'$ in the same bucket arrive, then we can add relatively short “connectors” to N so that it also contains an $a'b'$ -path of weight at most $(1 + \varepsilon)\|a'b'\|$. Thus N can easily accommodate new paths in the online model. The key technical tool for constructing Steiner networks N (one for each bucket) is the so-called *shallow-light trees*, introduced by Awerbuch et al. [7] and Khuller et al. [41], and optimized in the geometric setting by Elkin and Solomon [28, 52].

As a counterpart, we show (Theorem 7 in Section 4) that the dependence on n cannot be eliminated in dimensions $d \geq 2$. In particular, we prove that any $(1 + \varepsilon)$ -spanner for a sequence of n points in \mathbb{R}^d , has competitive ratio $\Omega(f(n))$ for some function $f(n)$ with $\lim_{n \rightarrow \infty} f(n) = \infty$. The lower bound construction consists of an adaptive strategy for the adversary in the plane: The adversary recursively maintains a space partition and places points in *rounds* so that the spanner constructed so far is disjoint from most of the ellipses B_{ab} that will contain the ab -paths for pairs of new points a, b . In order to control OPT, the adversary maintains the property that OPT_i is an x -monotone path γ_i after round i . However, this requirement means that any new point must be very close to γ_i , and S will be a set of almost collinear points. The core challenge of the Steiner spanner problem seems to lie in the case of almost collinear points.

Higher dimensions under the L_1 -norm. Finally, in the full version of our paper we provide improved lower bounds for points in \mathbb{R}^d under the L_1 norm (without Steiner points). We show that for every $\varepsilon > 0$, under the L_1 norm, the competitive ratio of any online $(1 + \varepsilon)$ -spanner algorithm $\Omega(\varepsilon^{-2}/\log \varepsilon^{-1})$ in \mathbb{R}^2 and is $\Omega(\varepsilon^{-d})$ in \mathbb{R}^d for $d \geq 3$.

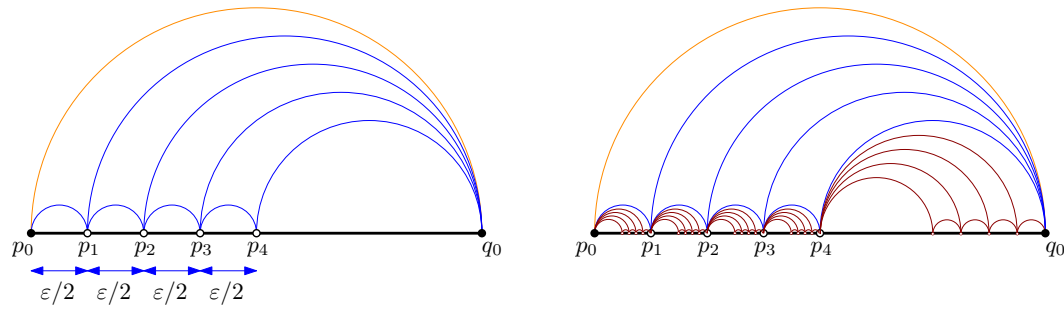
The adversary takes advantage of the non-monotonicity of OPT, mentioned above. In round 1, it presents a point set $S_1 \cup S_2$ for which any $(1 + \varepsilon)$ -spanner (without Steiner points) must contain a complete bipartite graph between S_1 and S_2 ; however the optimal Steiner $(1 + \varepsilon)$ -spanner for $S_1 \cup S_2$ has much smaller weight. Then in round 2, the adversary presents all Steiner points $\hat{S}_1 \cup \hat{S}_2$ of an optimal Steiner $(1 + \varepsilon)$ -spanner for $S_1 \cup S_2$. The key insight is that under the L_1 -norm (and for this particular point set), the optimal Steiner spanner for $S_1 \cup S_2$ already contains Manhattan paths between any two points in $S = (S_1 \cup S_2) \cup (\hat{S}_1 \cup \hat{S}_2)$, and so it remains the optimum solution (without Steiner points) for the point set S .

We were unable to replicate this phenomenon under the L_2 -norm, where the current best lower bound in \mathbb{R}^d , for all $d \geq 1$, derives from the 1-dimensional construction. In particular, it is not sufficient to consider the *Steiner ratio* for $(1 + \varepsilon)$ -spanners, defined as the supremum ratio between the weight of the minimum $(1 + \varepsilon)$ -spanner and the minimum Steiner $(1 + \varepsilon)$ -spanner of a finite point set in \mathbb{R}^d . Under the L_2 -norm, this ratio is $\Theta(\varepsilon^{-1})$ in the plane and $\tilde{\Theta}(\varepsilon^{(1-d)/2})$ in \mathbb{R}^d for $d \geq 3$ [12, 42, 44]. However, an optimal Steiner $(1 + \varepsilon)$ -spanner, need not achieve the desired $1 + \varepsilon$ stretch factor for the Steiner points.

2 Lower and Upper Bounds for Points on a Line

It is easy to analyze the one-dimensional case as the offline optimum network (OPT) for any set of points in a line is a path from the leftmost point to the rightmost point; the stretch factor of this path is always 1. (In contrast, in 2- and higher dimensions, the optimum $(1 + \varepsilon)$ -spanner is highly dependent on the distribution of points, which in turn may change over time in the online model.)

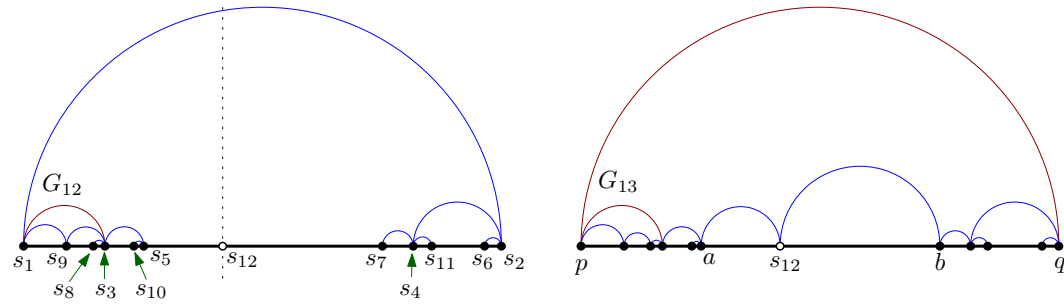
Lower bound. The following adversarial strategy establishes a lower bound $L(n) = \Omega(\varepsilon^{-1})$ for the competitive ratio; refer to Fig. 2 (left). Start with two points $p_0 = 0$ and $q_0 = 1$. For the first two points, ALG must add a direct edge p_0q_0 . Then the adversary successively places points $p_i = i \cdot \frac{\varepsilon}{2}$, for $i = 1, \dots, n$ so that all points remain in the interval $[0, \frac{1}{2}]$. Thus the number of points is $n = 2 + \lfloor \varepsilon^{-1} \rfloor$. In each round, ALG must add the edge p_iq_0 , otherwise any path between p_i and q_0 would have to make a detour via a point in $\{p_0, \dots, p_{i-1}\}$, and so it would be longer than $(1 + \varepsilon)\|p_iq_0\|$. Since $\|p_iq_0\| \geq \frac{1}{2}$, the weight of the network after $n - 2$ iterations is at least $\text{ALG} \geq 1 + \frac{1}{2}(n - 2) \geq 1 + \frac{1}{2}\lfloor \varepsilon^{-1} \rfloor$. Combined with $\text{OPT} = 1$, this yields a lower bound of $\Omega(\varepsilon^{-1})$ for the competitive ratio.



■ **Figure 2** Left: A sequence of n points $(q_0, p_0, p_1, p_2, \dots, p_{n-2})$, for $n = \lceil \varepsilon^{-1} \rceil$, for which any online $(1 + \varepsilon)$ -spanner has weight $\Omega(\varepsilon^{-1} \text{OPT})$. For clarity, the edges are drawn as circular arcs, but the weight of an edge $p_i p_j$ is $\|p_i p_j\| = |p_i - p_j|$. Right: Iteration in each subinterval.

The adversary has placed only $O(\varepsilon^{-1})$ points so far; this is the first stage of the strategy. In subsequent stages, the adversary repeats the same strategy in every subinterval ab of previous stage, as indicated in Fig. 2 (right). After stage $j \geq 1$, we have $\text{ALG} \geq 1 + \frac{j}{2} \lfloor \varepsilon^{-1} \rfloor = \Omega(j \varepsilon^{-1})$ and $\text{OPT} = 1$. The number of points placed in each stage increases by a factor of $\Omega(\varepsilon^{-1})$, hence $j = \Theta(\log_{\varepsilon^{-1}} n) = \Theta(\log n / \log \varepsilon^{-1})$. Overall, the competitive ratio is at least $\text{ALG}/\text{OPT} \geq \Omega(j \varepsilon^{-1}) = \Omega(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$.

Upper bound. For proving a matching upper bound in one-dimension, we use the following online algorithm: For all $i = 1, \dots, n$, we maintain a spanning graph G_i on $S_i = \{s_1, \dots, s_i\}$ and the x -monotone path P_i between the leftmost and the rightmost points in $S_i = \{s_1, \dots, s_i\}$. When point s_i , $i \geq 2$, arrives, we proceed as follows (see Fig. 3). If s_i is left (resp., right) of all previous points, we add an edge from s_i to the closest point in S_{i-1} to both P_{i-1} and G_{i-1} . Otherwise, let ab be the (unique) edge of P_{i-1} that contains s_i , and pq a shortest edge of G_{i-1} that contains s_i . Clearly, we have $P_i = P_{i-1} - ab + as_i + s_i b$. If $\min\{\|ps_i\|, \|s_i q\|\} > \frac{\varepsilon}{4} \|pq\|$, we add both as_i and $s_i b$ to G_i , that is, $G_i = G_{i-1} + as_i + s_i b$. Otherwise, let $G_i = G_{i-1} + as_i$ if $\|ps_i\| \leq \|s_i q\|$, or else $G_i = G_{i-1} + s_i b$.



■ **Figure 3** Left: The graph G_{12} for (s_1, \dots, s_{11}) . Right: $pq = s_1 s_2$ is the shortest edge of G_{11} that contain s_{12} . The algorithm adds edges $as_{12} = s_5 s_{12}$ and $s_{12} b = s_{12} s_7$.

We observe a few properties of G_i that are immediate from the construction: (P1) At the time when edge e is added to G_i , then the interior of e does not contain any vertices. (P2) The edges in G_i form a laminar set of intervals (i.e., any two edges are interior-disjoint, or one contains the other). (P3) If e_1, e_2 are edges in G_i and $e_2 \subset e_1$, then $\|e_2\| \leq (1 - \frac{\varepsilon}{4})\|e_1\|$. We note that properties (P1)–(P3) are inherently 1-dimensional, as the edges are intervals in \mathbb{R} , and they do not seem to generalize to higher dimensions.

► **Lemma 1.** For $i = 1, \dots, n$, the graph G_i is a $(1 + \varepsilon)$ -spanner for S_i .

The standard proof (by induction on n) is available in the full version of the paper.

► **Lemma 2.** For $i = 1, \dots, n$, we have $\|G_i\| \leq O(\varepsilon^{-1} \text{OPT}_i \log i / \log \varepsilon^{-1})$.

Proof. We may assume w.l.o.g. that $i = n$, and let $\text{OPT} = \text{OPT}_n$ for brevity. Let E be the edge set of G_n . The order in which ALG adds edges to E defines a (precedence) poset on E . We partition E by weight as follows: Let $\beta = \varepsilon^{-1}$; and for all $\ell \in \mathbb{Z}$, let E_ℓ be the set of edges $e \in E$ with $\beta^\ell < \|e\| \leq \beta^{\ell+1}$. Since $\|e\| \leq \text{OPT}$ for all $e \in E$, every edge is in E_ℓ for some $\ell \leq \log_\beta \text{OPT}$. Furthermore, for all $\ell \leq \log_\beta (\text{OPT}/n^2)$, the edges $e \in E_\ell$ have weight $\|e\| \leq \text{OPT}/n^2$, and so the total weight of these edges is less than OPT . It remains to consider E_ℓ for $\log_\beta (\text{OPT}/n^2) \leq \ell \leq \log_\beta \text{OPT}$, that is, for $O(\log n / \log \varepsilon^{-1})$ values of ℓ .

Let pq be an edge in E_ℓ that is not contained in any previous edge in E_ℓ . By property (P2), the edges in E_ℓ form a laminar family, and so pq does not overlap with any previous edge in E_ℓ ; and pq contains any subsequent edge that overlaps with it. Let $E_\ell(pq)$ be the set of all edges in E_ℓ that are contained in pq (including pq). We claim that

$$\|E_\ell(pq)\| \leq O(\varepsilon^{-1} \|pq\|). \quad (1)$$

Summation over all edges $pq \in E_\ell$ that are not contained in previous edges in E_ℓ implies $\|E_\ell\| \leq O(\varepsilon^{-1} \text{OPT})$. Summation over all $\ell \in \mathbb{Z}$ then yields

$$\|E\| = \sum_{\ell \in \mathbb{Z}} \|E_\ell\| = \sum_{\ell = \lfloor \log_\beta (\text{OPT}/n^2) \rfloor}^{\lceil \log_\beta \text{OPT} \rceil} \|E_\ell\| + O(\text{OPT}) = O(\varepsilon^{-1} \text{OPT} \log_\beta n).$$

To prove (1), consider the containment poset of $E_\ell(pq)$. In fact, we represent the poset as a rooted binary tree T : The root corresponds to pq , and edges $e_1, e_2 \in E_\ell(pq)$ are in parent-child relation iff $e_2 \subset e_1$, and there is no edge $e' \in E_\ell(pq)$ with $e_2 \subset e' \subset e_1$. Each level of T corresponds to interior-disjoint edges contained in $\|pq\|$, so the sum of weight on each level is at most $\|pq\|$. The total weight of the first $k = \lceil 5\varepsilon^{-1} \rceil$ levels is $O(\varepsilon^{-1} \|pq\|)$.

We claim that the total weight on level $k = \lceil 5\varepsilon^{-1} \rceil$ is at most $\frac{1}{2} \|pq\|$. We distinguish between three types of nodes in the subtree of T between levels 0 and k : A *branching node* has two children, a *single-child node* has one child, and a *leaf* has no children (in particular all nodes in level k are considered leaves in this subtree). The nodes (leaves) at level k correspond to interior-disjoint edges $e \subset pq$ with $\|e\| \geq \varepsilon \|pq\|$ by the definition of $E_\ell(pq)$. Thus there are at most $\lfloor \varepsilon^{-1} \rfloor$ nodes at level k , hence there are less than $\lfloor \varepsilon^{-1} \rfloor$ branching nodes. This implies that for any node e on level k , the descending path from the root pq to e contains at least $k - \lfloor \varepsilon^{-1} \rfloor \geq \lceil 4\varepsilon^{-1} \rceil$ single-child nodes.

For the purpose of bounding the total weight at level k , we can modify T , by incrementally moving all single-child nodes below all branching nodes as follows. While there is an edge uv in T , such that u is a branching node, and its parent v is a single-child node, we suppress u and subdivide the two edges of T below u with new nodes v_1 and v_2 . The weight along the edge uv goes down by a factor of at most $(1 - \frac{\varepsilon}{4})$ by property (P3); we set the weights in the modified tree such that the same decrease occurs along the edges uv_1 and uv_2 . Then each operation maintains property (P3), and the total weight at level k does not change. When the while loop terminates, we obtain a full binary tree with a chain attached to each leaf. As we argued above, each chain has length $\lfloor 4/\varepsilon \rfloor$ or more. The full binary tree does not necessarily decrease the weight. Along each chain of $\lfloor 4/\varepsilon \rfloor$ or more single-child nodes, the weight is cumulatively multiplied by a factor of at most $(1 - \frac{\varepsilon}{4})^{\lfloor 4/\varepsilon \rfloor} < \frac{1}{2}$. Overall, the total weight at level $k = \lceil 5\varepsilon^{-1} \rceil$ is at most $\frac{1}{2} \|pq\|$, as claimed.

By induction, for every integer $j \geq 0$, the total weight at level $jk = j\lceil 5\varepsilon^{-1} \rceil$ is at most $\|pq\|/2^j$. Consequently, the total weight of a *block* of k consecutive levels $\{jk+1, \dots, (j+1)k\}$ is at most $k\|pq\|/2^j$. Overall, $\|E_\ell(pq)\| = \sum_{j \geq 0} k\|pq\|/2^j = O(k\|pq\|) = O(\varepsilon^{-1}\|pq\|)$, which completes the proof of (1). \blacktriangleleft

We can summarize the discussion above in the following theorem.

► **Theorem 3.** *For every $\varepsilon > 0$, the competitive ratio of any online algorithm for $(1 + \varepsilon)$ -spanners for a sequence of points on a line is $\Omega(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$. Moreover, there is an online algorithm that maintains a $(1 + \varepsilon)$ -spanner with competitive ratio $O(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$.*

3 Upper Bounds for Spanners in \mathbb{R}^d under the L_2 Norm

We turn to online $(1 + \varepsilon)$ -spanners in Euclidean d -space for $d \geq 2$. The dynamic algorithm DEFSPANNER by Gao et al. [30], based on hierarchical clustering, achieves $O(\varepsilon^{-(d+1)} \log n)$ competitive ratio in the online model. In Section 3.1, we recover the same bound with a new analysis, where we refine the hierarchical clustering with a partition of the edges into buckets of similar directions, locations, and weights. In Section 3.2, we extend the new analysis to show that the competitive ratio improves to $O(\varepsilon^{(1-d)/2} \log n)$ if we are allowed to use Steiner points. Our spanner algorithm replaces each bucket of “similar” edges with a Steiner network using grids and shallow-light trees, for up to $O(\varepsilon^{(1-d)/2})$ directions.

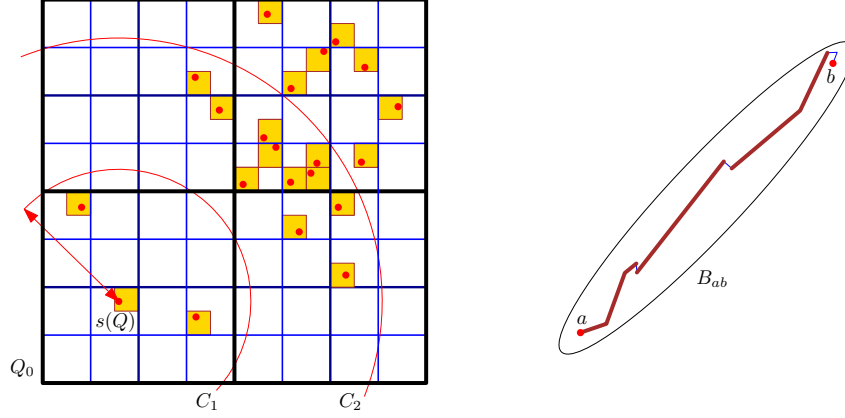
Preliminaries. *Well-separated pair-decomposition* (for short, WSPD) of a finite point set S in a metric space is a classical tool for constructing $(1 + \varepsilon)$ -spanners [19, 34, 47, 51]. It is a collection of pairs $\{(A_i, B_i) : i \in I\}$ such that for all $i \in I$, we have $A_i, B_i \subset S$ and $\max\{\text{diam}(A_i), \text{diam}(B_i)\} \leq \varepsilon \text{dist}(A_i, B_i)$; and for every point pair $\{s, t\} \subset \binom{S}{2}$, there is a pair (A_i, B_i) such that A_i and B_i each contains precisely one of s and t . It was shown by Callahan and Kosaraju [18] that if a graph $G = (S, E)$ contains an edge between arbitrary points in A_i and B_i , for all $i \in I$, then G is an $(1 + O(\varepsilon))$ -spanner for S ; see also [47, Ch. 9].

Dynamic spanners (including the fully dynamic algorithm by Roditty [49] and DEFSPANNER by Gao et al. [30]) rely on WSPDs and hierarchical clustering. In \mathbb{R}^d , hierarchical clustering can be obtained by classical recursive space partitions such as *quadtrees* [27, Ch. 14]. Dynamic quadtrees and their variants have been studied extensively, due to their broad range of applications; see [38, Ch. 2]. In general, dynamic quadtrees can handle both point insertion and deletion operations. However, in the context of an online algorithm, where the points are only inserted, note that no cell of the quadtree is ever deleted. We analyse the competitive ratio of the dynamic incremental algorithm by Fischer and Har-Peled [29] that maintains an $(1 + \varepsilon)$ -spanner for n points in Euclidean d -space in expected $O([\log n + \log \varepsilon^{-1}] \varepsilon^{-d} \log n)$ update time. However, they have not analyzed the ratio between the *weight* of the resulting $(1 + \varepsilon)$ -spanner and the minimum weight of an $(1 + \varepsilon)$ -spanner.

3.1 Online Algorithm without Steiner Points

Online Algorithm. We briefly review the algorithm in [29] and then analyze the weight. The input is a sequence of points (s_1, s_2, \dots) in \mathbb{R}^d ; the set of the first n points is denoted by $S_n = \{s_i : 1 \leq i \leq n\}$. For every n , we dynamically maintain a quadtree \mathcal{T}_n for S_n . Every node of \mathcal{T}_n corresponds to a cube. The root of \mathcal{T}_n , at level 0, corresponds to a cube Q_0 of side length $a_0 = \Theta(\text{diam}(S_n))$. At every level $\ell \geq 0$, there are at most $2^{d\ell}$ interior-disjoint cubes, each of side length $a_0/2^\ell$. A cube $Q \in \mathcal{T}_n$ is *nonempty* if $Q \cap S_n \neq \emptyset$. For every nonempty cube Q , we select an arbitrary representative $s(Q) \in Q \cap S_n$. At each level ℓ ,

let E_ℓ be the set of all edges $s(Q_1)s(Q_2)$ for pairs of cubes $\{Q_1, Q_2\}$ on level ℓ such that $\frac{c_1 a_0}{\varepsilon 2^\ell} \leq \|s(Q_1)s(Q_2)\| \leq \frac{c_2 a_0}{\varepsilon 2^\ell}$ for some constants $0 < c_1 < c_2$ that depend on d ; see Fig. 4(left). The algorithm maintains the spanner $G = (S_n, E)$ where $E = \bigcup_{\ell \geq 0} E_\ell$. A classical argument by Callahan and Kosaraju [18] (see also [34, 47, 51]) shows that G is a $(1 + \varepsilon)$ -spanner for S_n .



■ **Figure 4** Left: Nonempty squares at level $\ell = 4$ of a quadtree, each with a representative (red dots). Point $s(Q)$ is connected to all other representatives in the annulus between the concentric circles C_1 and C_2 of radii $c_1/(\varepsilon 2^\ell)$ and $c_2/(\varepsilon 2^\ell)$. Right: Ellipse B_{ab} with foci a and b , an ab -path of weight $(1 + \varepsilon)\|ab\|$. The bold edges make an angle at most $\varepsilon^{1/2}$ with ab .

► **Theorem 4.** For every constant $d \geq 2$, parameter $\varepsilon > 0$, and a sequence of $n \in \mathbb{N}$ points in Euclidean d -space, the competitive ratio of the online algorithm above is in $O(\varepsilon^{-(d+1)} \log n)$.

Proof. For the set S_n of the first n points of a sequence in \mathbb{R}^d , let $G = (S_n, E)$ be the $(1 + \varepsilon)$ -spanner produced by the online algorithm, and let $G^* = (S_n, E^*)$ be an $(1 + \varepsilon)$ -spanner of minimum weight. We show that $\|G\|/\|G^*\| = O(\varepsilon^{-(d+1)} \log n)$.

Short edges. Note that the weight of every edge in $E_\ell \subset E$ at level ℓ is $\Theta(\varepsilon^{-1} \text{diam}(S_n)/2^\ell)$, since it connects representatives at $\Theta(\varepsilon^{-1} \text{diam}(S_n)/2^\ell)$ distance apart. In particular, an edge at any level $\ell \geq 2 \log n$ has weight at most $O(\varepsilon^{-1} \text{diam}(S_n)/n^2)$; and the total weight of these edges is $O(\varepsilon^{-1} \text{diam}(S_n)) \leq O(\varepsilon^{-1} \text{OPT})$. It remains to bound the weight of the edges on levels $\ell = 1, \dots, \lfloor 2 \log n \rfloor$. We consider each level separately.

Short edges. For every edge $ab \in E$, let B_{ab} denote the ellipsoid with foci a and b , and great axis of length $(1 + \varepsilon)\|ab\|$. Note that every ab -path of weight at most $(1 + \varepsilon)\|ab\|$ lies in B_{ab} . The set of directions of line segments in \mathbb{R}^d is represented by a hemisphere of \mathbb{S}^{d-1} . The distance between two directions is measured by angles in the range $[0, \pi)$. Recently, Bhore and Tóth [13, Lemma 3] proved that every ab -path of weight at most $(1 + \varepsilon)\|ab\|$ contains edges of total weight at least $\frac{1}{2}\|ab\|$ that make an angle at most $\varepsilon^{1/2}$ with ab (i.e., they are near-parallel to ab); see Fig. 4(right).

Since G^* is a $(1 + \varepsilon)$ -spanner for S_n , it contains an ab -path of weight at most $(1 + \varepsilon)\|ab\|$ for every $ab \in E$. This path lies in the ellipsoid B_{ab} , and contains edges of G^* of weight at least $\frac{1}{2}\|ab\|$ and with direction with at most $\varepsilon^{1/2}$ from ab . We next define suitable *disjoint* sets of ellipsoids, in order to establish a lower bound on $\|G^*\|$.

Edge partition by directions. First, we partition the edge set E_ℓ into subsets based on the *directions* of the edges. We use standard volume argument to construct a *homogeneous* set of directions. Let $H \subset \mathbb{S}^{d-1}$ be the hemisphere of unit vectors in \mathbb{R}^d , then the direction vector of a line segment ab , denoted $\text{dir}(ab)$, is a unique point in H . Consider a maximal packing of H with (spherical) balls of radius $\frac{1}{8}\varepsilon^{1/2}$. Since the spherical volume of H is $\Theta(1)$ and the volume of each ball is $\Theta(\varepsilon^{(d-1)/2})$, the number of balls is $K = \Theta(\varepsilon^{(1-d)/2})$.

By doubling the radii of the spherical balls to $\frac{1}{4}\varepsilon^{1/2}$, we obtain a covering of H with a set of balls $\mathcal{D} = \{D_i : i = 1, \dots, K\}$. For each spherical ball $D_i \in \mathcal{D}$, denote by $2D_i$ the concentric ball of radius $\frac{1}{2}\varepsilon^{1/2}$. By standard packing argument, the ball $2D_i$ intersects only $O(1)$ balls in \mathcal{D} (where $d = O(1)$). We can now define a partition $E_\ell = \bigcup_{i=1}^K E_{\ell,i}$ as follows: let an $ab \in E_\ell$ be in $E_{\ell,i}$ if i is the smallest index such that $\text{dir}(ab) \in D_i$. Now for every $i = 1, \dots, K$, let E_i^* be the set of edges $e^* \in E^*$ such that $\text{dir}(e^*) \in 2D_i$. By construction, every edge $e^* \in E^*$ lies in $O(1)$ sets E_i^* ; consequently $\sum_{i=1}^K \|E_i^*\| = \Theta(\|G^*\|)$. Furthermore, for every edge $ab \in E_{\ell,i}$, all edges in E^* that make an angle at most $\varepsilon^{1/2}$ with ab are in E_i^* .

Disjoint ellipsoids. For every $i = 1, \dots, K$, let $\mathcal{B}_{\ell,i}$ be the set of ellipsoids B_{ab} with $ab \in E_{\ell,i}$. We show that $\mathcal{B}_{\ell,i}$ contains a subset $\mathcal{B}'_{\ell,i}$ of disjoint ellipsoids such that $|\mathcal{B}'_{\ell,i}| \geq \Omega(\varepsilon^{d+1}|\mathcal{B}_{\ell,i}|)$.

We claim that every ellipsoid in $\mathcal{B}_{\ell,i}$ intersects $O(\varepsilon^{-(d+1)})$ other ellipsoids in $\mathcal{B}_{\ell,i}$. We make use of a volume argument. Let $M_\ell = \max\{\|e\| : e \in E_\ell\}$; and note that the side length of every cube at level ℓ of the quadtree is $\Theta(\varepsilon M_\ell)$.

For every ellipsoid $B_{ab} \in \mathcal{B}_{\ell,i}$, the great axis has length $(1 + \varepsilon)\|ab\|$, and the $d - 1$ minor axes each have length $\sqrt{(1 + \varepsilon)^2 - 1^2}\|ab\| < 2\varepsilon^{1/2}\|ab\|$, where $\|ab\| \leq M_\ell$. Hence B_{ab} is contained in a cylinder C_{ab} of height $(1 + \varepsilon)M_\ell$ whose base is a $(d - 1)$ -dimensional ball of diameter $2\varepsilon^{1/2}M_\ell$. Any other ellipsoid in $\mathcal{B}_{\ell,i}$ with great axis parallel to ab is contained in a translate of C_{ab} . If we rotate B_{ab} about its center by an angle at most $\varepsilon^{1/2}$, then its orthogonal projection to the original great axis decreases, and the maximum distance from the original great axis increases by at most $\|ab\| \frac{1+\varepsilon}{2} \sin \varepsilon^{1/2} < M_\ell \varepsilon^{1/2}$. Consequently, every ellipsoid in $\mathcal{B}_{\ell,i}$ is contained in a translated copy of $2C_{ab}$. Hence, every ellipsoid in $\mathcal{B}_{\ell,i}$ that intersects B_{ab} is contained in $3C_{ab}$. Every cube at level ℓ of the quadtree that intersects $3C_{ab}$ is contained in the Minkowski sum of $3C_{ab}$ and such a cube, which is in turn contained in $4C_{ab}$. Note that the volume of the cylinder $4C_{ab}$ is $O(\varepsilon^{(d-1)/2}M_\ell^d)$; while the volume of a cube at level ℓ of the quadtree is $\Theta(\varepsilon^d M_\ell^d)$. Therefore $4C_{ab}$ contains $O(\varepsilon^{(d-1)/2}/\varepsilon^d) = O(\varepsilon^{-(d+1)/2})$ such cubes. Recall that the algorithm maintains one representative from each cube, and the edges $ab \in E_{\ell,i}$ are pairs of representative. Thus $O(\varepsilon^{-(d+1)/2})$ representatives in $4C_{ab}$ can form $O(\varepsilon^{-(d+1)})$ pairs (i.e., edges, hence ellipsoids).

This completes the proof of the claim that every ellipsoid in $\mathcal{B}_{\ell,i}$ intersects $O(\varepsilon^{-(d+1)})$ other ellipsoids in $\mathcal{B}_{\ell,i}$. Hence the *intersection graph* of $\mathcal{B}_{\ell,i}$ is $O(\varepsilon^{-(d+1)})$ -degenerate; and has an independent set $\mathcal{B}'_{\ell,i}$ of size $|\mathcal{B}'_{\ell,i}| \geq \Omega(\varepsilon^{d+1}|\mathcal{B}_{\ell,i}|) = \Omega(\varepsilon^{d+1}|E_{\ell,i}|)$.

Weight analysis. As noted above, all edges in E_ℓ have length $\Theta(M_\ell)$. For every $i = 1, \dots, K$ and for every ellipsoid $B_{ab} \in \mathcal{B}_{\ell,i}$, we have $\|E_i^* \cap B_{ab}\| \geq \frac{1}{2}\|ab\| \Omega(M_\ell)$. Summing over a set of disjoint ellipsoids, we obtain

$$\begin{aligned} \|E_i^*\| &\geq \sum_{B_{ab} \in \mathcal{B}'_{\ell,i}} \|E_i^* \cap B_{ab}\| \geq \sum_{B_{ab} \in \mathcal{B}'_{\ell,i}} \frac{1}{2}\|ab\| \\ &\geq |\mathcal{B}'_{\ell,i}| \cdot \frac{1}{2} \min\{\|ab\| : ab \in E_{\ell,i}\} \\ &\geq \varepsilon^{-(d+1)}|E_{\ell,i}| \cdot \Omega(M_\ell) = \Omega(\varepsilon^{-(d+1)}\|E_{\ell,i}\|). \end{aligned}$$

Summation over all directions $i = 1, \dots, K$ yields

$$\|G^*\| = \Theta \left(\sum_{i=1}^K \|E_i^*\| \right) \geq \Omega \left(\sum_{i=1}^K \varepsilon^{-(d+1)} \|E_{\ell,i}\| \right) = \Omega(\varepsilon^{-(d+1)} \|E_\ell\|).$$

Finally, summation over all $\ell \geq 1$ yields

$$\|E\| = \sum_{\ell \geq 1} \|E_\ell\| \leq \sum_{\ell=1}^{\lfloor 2 \log n \rfloor} \|E_\ell\| + \sum_{\ell > \lfloor 2 \log n \rfloor} \|E_\ell\| \leq \varepsilon^{-(d+1)} \|G^*\| \log n + \varepsilon^{-1} \|G^*\|,$$

as required. \blacktriangleleft

3.2 Online Algorithm with Steiner Points

When Steiner points are allowed, we can substantially improve the competitive ratio in terms of ε . We describe an algorithm with competitive ratio $O(\varepsilon^{(1-d)/2} \log n)$. As a counterpart, we show in Section 4 that the dependence on n is unavoidable in dimensions $d \geq 2$; it remains an open problem whether the dependence on ε is necessary.

► **Theorem 5.** *For every $\varepsilon > 0$, an online algorithm can maintain, for a sequence of $n \in \mathbb{N}$ points in the plane, a Euclidean Steiner $(1 + \varepsilon)$ -spanner of weight $O(\varepsilon^{-1/2} \log n) \cdot \text{OPT}$.*

Proof. Our online algorithm has two *stages*: A_1 and A_2 . Algorithm A_1 is the same as in Section 3.1, it maintains a quadtree \mathcal{T}_n for the point set S_n , and a “primary” $(1 + \varepsilon)$ -spanner G_1 *without* Steiner points. Algorithm A_2 maintains a Steiner $(1 + 3\varepsilon)$ -spanner G_2 as follows: for each edge ab in G_1 , it creates an ab -path of length $(1 + \varepsilon)\|ab\|$ using Steiner points in G_2 . Importantly, algorithm A_2 can bundle together “similar” edges of G_1 , and handle them together using shallow-light trees [52].

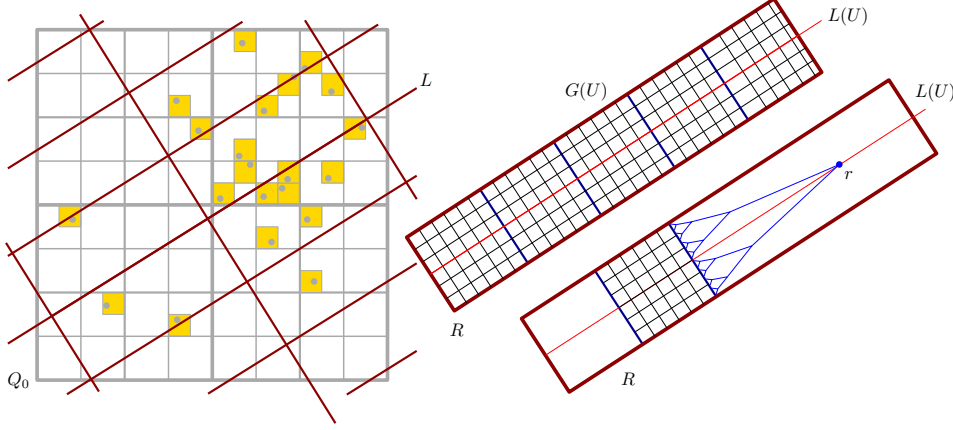
In particular, we partition the space of all possible edges of G_1 into *buckets* (edges with similar directions, locations, and weights). For each bucket U , when algorithm A_1 inserts the first edge $ab \in U$ into G_1 , then algorithm A_2 creates a “backbone” Steiner tree $T = T(U)$ of weight $O(\|ab\|)$, which contains an ab -path of length at most $(1 + \varepsilon)\|ab\|$. For any subsequent edge $a'b' \in U$, it suffices to add paths from a' and b' to T , of weight $O(\varepsilon\|ab\|)$, to obtain $a'b'$ -path of length at most $(1 + \varepsilon)\|a'b'\|$. Overall, between any two points $s_i, s_j \in S$, the primary spanner contains a path of weight at most $(1 + \varepsilon)\|s_i s_j\|$, and G_2 contains an Steiner path of weight at most $(1 + \varepsilon)^2\|s_i s_j\| < (1 + 3\varepsilon)\|s_i s_j\|$, as claimed.

It remains to define the buckets U , the backbone $T(U)$ for the first edge in U , and the “connectors” added for each subsequent edge in U . We first describe the algorithm in the plane, where we establish a competitive ratio $O(\varepsilon^{-1/2} \log n)$, and then generalize the construction to higher dimensions.

Buckets. We define buckets for all potential edges in the primary spanner G_1 . We analyze a single level ℓ of the quadtree \mathcal{T} . Without loss of generality, assume that the side length of all quadtree cubes in level ℓ have unit length, hence the weight of every edge in E_ℓ is $\Theta(\varepsilon^{-1})$.

In Section 3.2, we have covered the set $H \subset \mathbb{S}^1$ of directions with a set $\mathcal{D} = \{D_i : i = 1, \dots, K\}$ of balls of diameter $\varepsilon^{1/2}$. For each ball in \mathcal{D} , we define a set of buckets. Let $D \in \mathcal{D}$, and let L be a line such that $\text{dir}(L)$ corresponds to the center of D ; refer to Fig. 5(left). Partition the plane into parallel strips of width $\frac{1}{2}\varepsilon^{1/2}$ by a set of lines parallel to L ; and partition each strip further into rectangles of height $2\varepsilon^{-1}$. By scaling up the rectangles by a factor of 2, we obtain a covering of the square Q_0 with a set \mathcal{R} of $4\varepsilon^{-1} \times \varepsilon^{1/2}$ rectangles such that each point is covered by $O(1)$ rectangles in \mathcal{R} .

For each rectangle $R \in \mathcal{R}$, we create a bucket U comprising all edges $ab \in E_\ell$ such that $ab \subset R$ and $\text{dir}(ab) \in D$ (hence $\angle(\text{dir}(ab), \text{dir}(L)) \leq \varepsilon^{1/2}$). Note that every edge $ab \in E_\ell$ lies in at least one and at most $O(1)$ buckets.



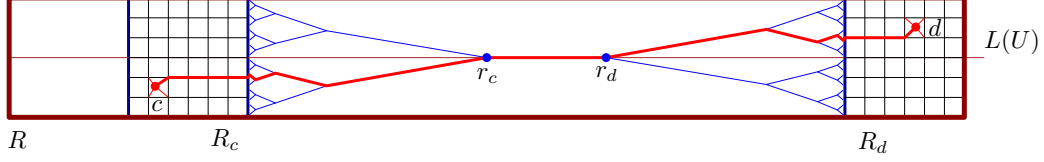
■ **Figure 5** Left: The overlay the the quadtree with a partition of \mathbb{R}^2 into $\frac{1}{2}\varepsilon^{-1/2} \times 2\varepsilon^{-1}$ rectangles aligned with L . Top-Right: A rectangle $R \in \mathcal{R}$, the median $L(U)$, the grid $G(U)$, and the partition of R into $\varepsilon^{-1/2} \times \varepsilon^{-1/2}$ squares. Bottom-Right: A shallow-light tree between a side of an $\frac{1}{2}\varepsilon^{-1/2} \times \frac{1}{2}\varepsilon^{-1/2}$ square and a source $r \in L(U)$.

Backbones and Connectors. Let U be a bucket defined above for a rectangle $R \in \mathcal{R}$. Let $L(U)$ denote the median of the rectangle R parallel to L . When the primary algorithm A_1 inserts the first edge $ab \in U$ into G_1 , then Algorithm A_2 constructs a unit grid graph $G(U)$, formed by a subdivision of R into unit squares; see Fig. 5(top-right). Since R is a $4\varepsilon^{-1} \times \varepsilon^{-1/2}$ rectangle, $\|G(U)\| = O(\varepsilon^{-3/2})$. Furthermore, we partition R into $4\varepsilon^{-1/2}$ squares of side length $\varepsilon^{-1/2}$. For each such square, we insert two shallow-light trees [52] between the two sides of the square orthogonal to L and two points in $L(U)$ at distance ε^{-1} from the square on either side; Fig. 5(bottom-right). The weight of each shallow-light tree is $O(\varepsilon^{-1})$ [52], and so the combined weight of $O(\varepsilon^{-1/2})$ shallow-light trees is $O(\varepsilon^{-3/2})$. The grid $G(U)$ together with the shallow-light trees forms the *backbone* for the bucket U in G_2 .

We add *connector* edges between a (resp., b) and the four corners of unit square of the grid $G(U)$ that contains it. For any subsequent edge $a'b' \in U$ that algorithm A_1 inserts into G_1 , the backbone does not change, we only add connectors between a' (resp., b') and the four corners of the unit square in $G(U)$ that contains it. The weight of the four connectors is $O(1)$ per point. Since $\text{area}(R) = \Theta(\varepsilon^{-3/2})$, then R intersects at most $O(\varepsilon^{-3/2})$ unit squares of the quadtree at level ℓ , and so the total weight of all connectors is $O(\varepsilon^{-3/2})$, as well.

Stretch analysis. Suppose algorithm A_1 inserts an edge cd into G_1 . As noted above, cd lies in $\Theta(1)$ buckets; refer to Fig. 6. Suppose bucket U contains cd ; and in the partition of the rectangle $R = R(U)$, the endpoint c (d) lies squares R_c (R_d) of side length $\varepsilon^{-1/2}$, associated with shallow-light trees rooted at r_c (r_d). Then G_2 contains a cd -path comprised of: (i) connectors from c and d , resp., to the closest point in the grid $G(U)$; (ii) paths in $G(U)$ from the connectors to the boundary of squares R_c and R_d , (iii) paths along the shallow-light trees to the roots $r_c, r_d \in L(U)$, and (iv) the line segment $r_c r_d$ in $G(U)$. The weight of each connector in (i) is at most $2\sqrt{2}$, which is bounded by $O(\varepsilon)\|cd\|$ since $\|cd\| = \Theta(\varepsilon^{-1})$. The edges in (ii) and (iv) are parallel to L , hence they make an angle less than $\varepsilon^{1/2}$ with

cd . Finally, consider the two subpaths in part (iii) in shallow-light trees: The line segment between the two endpoints of each such subpath makes an angle less than $\varepsilon^{1/2}$ with L , hence less than $2\varepsilon^{1/2}$ with cd ; and the weight of a root-to-leaf path in a shallow-light tree is a $(1 + \varepsilon)$ -approximation of the straight-line segment between its endpoints. Overall, the total weight of the cd -path described above is $(1 + O(\varepsilon))\|cd\|$, as required.



■ **Figure 6** A cd -path in the Steiner spanner G_2 .

For every point pair $a, b \in S_n$, the primary graph G_1 contains an ab -path $P = (p_0, \dots, p_m)$ of length $\|P\| \leq (1 + \varepsilon)\|ab\|$, since G_1 is a $(1 + \varepsilon)$ -spanner. We have shown that for every edge $p_{i-1}p_i$ of G_1 , the Steiner spanner G_2 contains a $p_{i-1}p_i$ -path of weight $(1 + O(\varepsilon))\|p_{i-1}p_i\|$. The concatenation of these paths yields an ab -path in G_2 , of weight $\sum_{i=1}^m (1 + O(\varepsilon))\|p_{i-1}p_i\| = (1 + O(\varepsilon))\|P\| = (1 + O(\varepsilon))(1 + \varepsilon)\|ab\| = (1 + O(\varepsilon))\|ab\|$.

Competitive Analysis. Denote by E_ℓ the set of edges of G_2 added at level $\ell = 1, \dots, 2 \log n$, and let b_ℓ be the number of nonempty buckets at level ℓ . We have seen that for each nonempty bucket at level ℓ , E_ℓ contains a subgraph of weight $O(\varepsilon^{-3/2} \text{diam}(S_n)/2^\ell)$; hence $\|E_\ell\| \leq O(b_\ell \cdot \varepsilon^{-3/2} \text{diam}(S_n)/2^\ell)$.

Let $G^* = (S_n, E^*)$ be a Euclidean Steiner $(1 + \varepsilon)$ -spanner for S_n of minimum weight OPT. Consider a nonempty bucket U associated with a line L and a rectangle $R(U)$. Since U is nonempty, there is an edge $ab \in U$ in G_1 . Recall that $ab \in R$ and $\angle(\text{dir}(ab), \text{dir}(L)) \leq \varepsilon^{1/2}$. Since G^* is a $(1 + \varepsilon)$ -spanner, it contains an ab -path P_{ab} of weight at most $(1 + \varepsilon)\|ab\|$. As noted in Section 3.1, P_{ab} lies in the ellipse B_{ab} , and contains edges of weight at least $\frac{1}{2}\|ab\|$ that make an angle at most $\varepsilon^{1/2}$ with ab . All points in the ellipse B_{ab} are at distance less than $\varepsilon^{1/2}$ from the line segment ab . The segment ab lies in the $4\varepsilon^{-1} \times \varepsilon^{1/2}$ rectangle $R(U)$. Thus we have $P_{ab} \subset B_{ab} \subset 2R(U)$, and so $2R(U)$ contains edges of G^* of weight $\frac{1}{2}\|ab\| = \Omega(\varepsilon^{-1} \text{diam}(S_n)/2^\ell)$ whose directions are within $2\varepsilon^{1/2}$ from L ; denote by $E^*(U) \subset E^*$ the set of these edges. By construction, each edge e^* of G^* lies in $E^*(U)$ for only $O(1)$ buckets. Indeed, there are $O(1)$ lines L' with $\angle(\text{dir}(L), \text{dir}(L')) \leq 2\varepsilon^{1/2}$, and for each such direction L' , every point in \mathbb{R}^2 lies in $O(1)$ rectangles $2R(U')$ aligned with L' . We conclude that $\text{OPT} = \|G^*\| = \Omega(b_\ell \cdot \varepsilon^{-1} \text{diam}(S_n)/2^\ell)$. This implies $\|E_\ell\|/\text{OPT} \leq O(\varepsilon^{-1/2})$ for $\ell = 1, \dots, 2 \log n$. Summation over all levels yields

$$\frac{\text{ALG}}{\text{OPT}} = \frac{\sum_{\ell=1}^{\infty} \|E_\ell\|}{\text{OPT}} \leq \sum_{\ell=1}^{2 \log n} O(\varepsilon^{-1/2}) + O(1) = O(\varepsilon^{-1/2} \log n),$$

as claimed. ◀

Generalization to \mathbb{R}^d . Our algorithm and its analysis generalize to Euclidean d -space.

► **Theorem 6.** *For every $\varepsilon > 0$, an online algorithm can maintain, for a sequence of $n \in \mathbb{N}$ points in \mathbb{R}^d , a Euclidean Steiner $(1 + \varepsilon)$ -spanner of weight $O(\varepsilon^{(1-d)/2} \log n) \cdot \text{OPT}$.*

The proof is analogous to that of Theorem 5. The bottleneck of the competitive analysis is the size of the unit grids $G(U)$ which is $\Theta(\varepsilon^{-(d+1)/2})$ in \mathbb{R}^d , and it is contrasted with a path of weight $\Omega(\varepsilon^{-1})$ in OPT. Similarly to Section 3.1, we choose a homogeneous set D of

$\Theta(\varepsilon^{(1-d)/2})$ directions. For each direction $L \in D$, we construct a tiling of \mathbb{R}^d with congruent hyper-rectangles aligned with L of dimensions $\varepsilon^{-1} \times \varepsilon^{-1/2} \times \dots \varepsilon^{-1/2}$. Refer to the full paper for all further details.

4 Lower Bound with Steiner Points

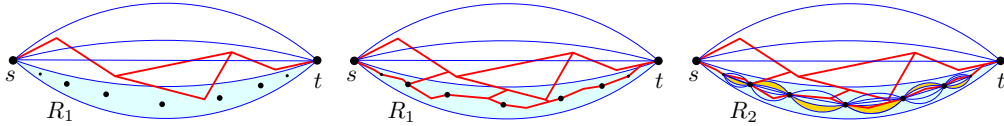
Recall that when Steiner points are allowed, the algorithm may subdivide existing edges with Steiner points. It follows that the in one-dimension, an online algorithm can maintain a Hamiltonian path on S_n , which is the minimum $(1 + \varepsilon)$ spanner for all $\varepsilon \geq 0$. This property carries over to Euclidean Steiner 1-spanners (i.e., the case $\varepsilon = 0$), where we need to maintain the complete straight-line graph on n points. However, we show that for $\varepsilon > 0$ in dimensions $d \geq 2$, the competitive ratio of an online algorithm with Steiner points must depend on n .

► **Theorem 7.** *For every $\varepsilon > 0$, the competitive ratio of any online algorithm that maintains a Euclidean Steiner $(1 + \varepsilon)$ -spanner for a sequence of n points in \mathbb{R}^d is $\Omega(f(n))$ for some function $f(n)$ such that $\lim_{n \rightarrow \infty} f(n) = \infty$.*

Proof. We describe and analyze an adversarial strategy for placing points in the plane in *stages*. In stage 1, the adversary places two points at $s = (0, 0)$ and $t = (1, 0)$, both on the x -axis. In subsequent stages, new points are arranged so that the optimum solution remains an x -monotone path of length at most $1 + \varepsilon$ at all times.

Let us denote by A_i the points placed in stage i . At the end of stage i , adversary constructs the point set A_{i+1} based on the current $(1 + \varepsilon)$ -spanner built by the algorithm ALG, and then placed the points in A_{i+1} in an arbitrary order. The objective is that in each stage, ALG has to add new edges of total weight at least $1/2$. Since $\text{OPT} \leq 1 + \varepsilon$ at all times, and $\text{ALG} \geq \frac{1}{2}(i + 1)$ after i stages, the competitive ratio goes to infinity.

We describe stage 2 in more detail; subsequent stages are similar; see Fig. 7. At the end of stage 1, our point set is $A_1 = \{s = (0, 0), t = (1, 0)\}$, the optimal spanner is a single edge of unit weight, and ALG has constructed a Euclidean Steiner $(1 + \varepsilon)$ -spanner G_1 for A_1 . Let $k_1 = \lceil \|G_1\| \rceil$. The adversary considers $2k_1 + 1$ circular arcs between s and t , each of weight at most $1 + \frac{\varepsilon}{2}$. The arcs define $2k_1$ interior-disjoint bounded regions. Let R_1 be a region that minimizes the weight $\|G_1 \cap R_1\|$, in particular, $\|G_1 \cap R_1\| \leq \frac{1}{2k_1} \|G_1\| \leq \frac{1}{2}$. In the interior of R_1 , let γ_1 be another circular arc between s and t , of weight $\|\gamma_1\| \leq 1 + \frac{\varepsilon}{2}$; and let $A_2 = \{t_1, \dots, t_N\}$ be a set of points along γ_1 , labeled in x -monotone increasing order with the following properties: (1) For every $i = 1, \dots, N - 1$, the ellipse B_i with foci t_i and t_{i+1} , and great axis $(1 + \varepsilon)\|t_i t_{i+1}\|$ lies entirely in R_1 ; and (2) the weight of the x -monotone path (t_1, t_2, \dots, t_N) is at least 1.



■ **Figure 7** Left: For $A_1 = \{s, t\}$, ALG constructs a $(1 + \varepsilon)$ -spanner G_1 (red). Five circular arcs define four regions; region R_1 satisfies $\|G_1 \cap R_1\| \leq \frac{1}{4} \|G_1\|$. In stage 2, the adversary presents points A_2 in R_1 . Middle: The algorithm augments G_1 to G_2 . Right: Region R_2 satisfies $\|G_2 \cap R_2\| \leq \frac{1}{2k_2} \|G_2\|$.

In stage 2, the adversary presents the points in A_2 in an arbitrary order. By the end of stage 2, ALG augments G_1 to a Euclidean Steiner $(1 + \varepsilon)$ -spanner G_2 for $A_1 \cup A_2$. In particular, for every $i = 1, \dots, N - 1$, the graph G_2 contains a $t_i t_{i+1}$ -path of length at most $(1 + \varepsilon)\|t_i t_{i+1}\|$, which lies in the ellipse E_i , hence in the interior of the region R_1 . The part of

the path between the vertical lines passing through t_i and t_{i+1} has weight at least $\|t_i t_{i+1}\|$. Since these parts are disjoint, the total weight all $N-1$ paths is at least $\sum_{i=1}^{N-1} \|t_i t_{i+1}\| \geq 1$. Consequently, $\|G_2 \cap R_1\| \geq 1$. Since we had $\|G_1 \cap R_1\| \leq \frac{1}{2}$, ALG must have added new edges of weight at least $\frac{1}{2}$ in stage 2, as claimed.

In phase $i+1$, in general, let $k_i = \lceil \|G_i\| \rceil$. Label the points in the current point set $S = \bigcup_{j=1}^i A_j$ by s_0, \dots, s_n in x -monotone order, and assume that the x -monotone path spanned by S has weight $\text{OPT} = 1 + (1 - \frac{1}{2^i})\varepsilon$. For all segments $s_j s_{j+1}$, we consider $2k_i + 1$ x -monotone circular arcs such that the total weight of any concatenation of the circular arcs from $s = s_0$ to $t = s_n$ is at most $1 + (1 - \frac{1}{2^{i+1}})\varepsilon$. For each segment $s_j s_{j+1}$, we choose one of $2k_i$ regions that has a minimum-weight intersection with G_i , and let R_i be the union of these regions. Note that $\|G_i \cap R_i\| \leq \frac{1}{2k_i} \|G_i\| \leq \frac{1}{2}$. Let γ_i be an st -path γ_i that connects the points s_0, \dots, s_n via circular arcs in the region R_i , and has weight at most $1 + (1 - \frac{1}{2^{i+1}})\varepsilon$. Now the adversary can choose a finite point set $A_{i+1} = \{t_1, \dots, t_N\}$ along γ_i with properties (1)–(2) above. This completes the description of the adversarial strategy.

Similarly to stage 2, when ALG augments G_i to a Euclidean Steiner $(1 + \varepsilon)$ -spanner G_{i+1} for $\bigcup_{j=1}^{i+1} A_j$, he must add new edges of weight at least $\frac{1}{2}$ in the region R_i . It follows that the competitive ratio for any online algorithm goes to infinity as n goes to infinity. \blacktriangleleft

5 Conclusions

We have studied online spanners for sequences of points in \mathbb{R}^d , in fixed dimensions $d \geq 1$, under L_2 and L_1 norms. We established a tight bound of $\Theta(\varepsilon^{-1} \log n / \log \varepsilon^{-1})$ for the competitive ratio of any online $(1 + \varepsilon)$ -spanner algorithms on a real line (Theorem 3). However it remains an open problem to close the gap between the lower and upper bounds in \mathbb{R}^d , for $d \geq 2$. Under the L_2 norm, previously known algorithms achieve competitive ratio $O(\varepsilon^{-(d+1)} \log n)$ (Theorem 4). The best lower bound we are aware of holds for $d = 1$. It is unclear whether the lower bound can be improved to $\varepsilon^{-\omega(d)} \log n$ for $d \geq 2$.

Next, we have showed that, if an online algorithm is allowed to use Steiner points, it can achieve a substantially better competitive ratio in terms of ε , namely $O(\varepsilon^{(1-d)/2} \log n)$, for a sequence of n points in \mathbb{R}^d and any constant $d \geq 2$, under the L_2 norm (Theorem 6). As a counterpart, we proved that any online spanner algorithm for a sequence of n points in \mathbb{R}^d under L_2 norm has competitive ratio $\Omega(f(n))$, where $\lim_{n \rightarrow \infty} f(n) = \infty$ (Theorem 7). It remains an open problem whether the competitive ratio depends on ε for Euclidean Steiner spanners. Another open problem is whether the factor $\log n$ in the upper bounds can be reduced, e.g., to $\log n / \log \log n$; similar to the work by Alon and Azar [2] who established such a lower bound for Euclidean minimum Steiner trees (EMST) for n points in \mathbb{R}^2 .

We have established a lower bound $\Omega(\varepsilon^{-d})$ for the competitive ratio under the L_1 -norm in \mathbb{R}^d . It is unclear whether it can be improved by a $\log n$ factor in dimensions $d \geq 2$. Designing online algorithms that match these bounds under the L_1 norm is left for future research.

In online spanner algorithms, the decisions are irrevocable, which means that once an edge is added to the spanner by an online algorithm, it can never be deleted. However, if some of the decisions are reversible, better bounds may be possible. This model is commonly known as *online algorithms with recourse* [33, 39, 45]. In 1-dimension, for instance, an optimum spanner is just a monotone path connecting the points in linear order, and any online algorithm that is allowed to remove at least one edge at per iteration can maintain such a path. In higher dimensions, however, it is unclear whether a $O(1)$ -approximation of the minimum-weight $(1 + \varepsilon)$ -spanner can be maintained with $O(\varepsilon^{-d+1})$ recourse.

References

- 1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.
- 2 Noga Alon and Yossi Azar. On-line Steiner trees in the Euclidean plane. *Discrete & Computational Geometry*, 10:113–121, 1993.
- 3 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.
- 4 Sunil Arya, David M Mount, and Michiel Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 1994.
- 5 Sunil Arya and Michiel Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997.
- 6 Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoretical Computer Science*, 324(2-3):313–324, 2004.
- 7 Baruch Awerbuch, Alan E. Baratz, and David Peleg. Cost-sensitive analysis of communication protocols. In *Proc. 9th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 177–187, 1990.
- 8 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012.
- 9 Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855, 2021.
- 10 Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems. In *Proc. 29th ACM Symposium on Theory of Computing (STOC)*, pages 344–353, 1997.
- 11 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1899–1918, 2019.
- 12 Sujoy Bhore and Csaba D. Tóth. Light Euclidean Steiner spanners in the plane. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, volume 189 of *LIPIcs*, pages 31:1–17. Schloss Dagstuhl, 2021.
- 13 Sujoy Bhore and Csaba D. Tóth. On Euclidean Steiner $(1+\varepsilon)$ -spanners. In *Proc. 38th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 187 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl, 2021.
- 14 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 15 Prosenjit Bose, Joachim Gudmundsson, and Pat Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004.
- 16 Prosenjit Bose and Michiel H. M. Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom.*, 46(7):818–830, 2013.
- 17 Paul B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 332–340, 1993.
- 18 Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In Vijaya Ramachandran, editor, *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 291–300, 1993.
- 19 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *J. ACM*, 42(1):67–90, 1995.
- 20 Paz Carmi and Lilach Chaitman-Yerushalmi. Minimum weight Euclidean t -spanner is NP-hard. *Journal of Discrete Algorithms*, 22:30–42, 2013.

- 21 Timothy M. Chan, Sariel Har-Peled, and Mitchell Jones. On locality-sensitive orderings and their applications. *SIAM J. Comput.*, 49(3):583–600, 2020.
- 22 L. Paul Chew. There is a planar graph almost as good as the complete graph. In *Proc. 2nd Symposium on Computational Geometry (SoCG)*, pages 169–177. ACM Press, 1986.
- 23 L. Paul Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, 1989.
- 24 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, pages 56–65, 1987.
- 25 Gautam Das, Paul Heffernan, and Giri Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Symposium on Computational Geometry (SoCG)*, pages 53–62. ACM Press, 1993.
- 26 Gautam Das, Giri Narasimhan, and Jeffrey S. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–222, 1995.
- 27 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3 edition, 2008.
- 28 Michael Elkin and Shay Solomon. Steiner shallow-light trees are exponentially lighter than spanning ones. *SIAM Journal on Computing*, 44(4):996–1025, 2015.
- 29 John Fischer and Sariel Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proc. 17th Canadian Conference on Computational Geometry (CCCG)*, pages 235–238, 2005.
- 30 Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006.
- 31 Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient regression in metric spaces via approximate Lipschitz extension. *IEEE Transactions on Information Theory*, 63(8):4838–4849, 2017.
- 32 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th European Symposium on Algorithms (ESA)*, volume 5193 of *LNCS*, pages 478–489. Springer, 2008.
- 33 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive Steiner tree online. *SIAM Journal on Computing*, 45(1):1–28, 2016.
- 34 Joachim Gudmundsson and Christian Knauer. Dilation and detours in geometric networks. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, volume 2. Chapman and Hall/CRC, 2nd edition, 2018.
- 35 Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002.
- 36 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms (TALG)*, 4(1):1–34, 2008.
- 37 Mohammad Taghi Hajiaghayi, Vahid Liaghat, and Debmalaya Panigrahi. Online node-weighted Steiner forest and extensions via disk paintings. In *Proc. 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 558–567, 2013.
- 38 Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. AMS, Providence, RI, 2011.
- 39 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.
- 40 J. Mark Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 318 of *LNCS*, pages 208–213. Springer, 1988.
- 41 Samir Khuller, Balaji Raghavachari, and Neal E. Young. Balancing minimum spanning and shortest path trees. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 243–250, 1993.
- 42 Hung Le and Shay Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100, 2019.

- 43 Hung Le and Shay Solomon. Light Euclidean spanners with Steiner points. In *Proc. 28th European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, pages 67:1–67:22. Schloss Dagstuhl, 2020.
- 44 Hung Le and Shay Solomon. A unified and fine-grained approach for light spanners. *CoRR*, abs/2008.10582, 2020. [arXiv:2008.10582](#).
- 45 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM Journal on Computing*, 45(3):859–880, 2016.
- 46 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted Steiner tree and related problems. In *Proc. 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 210–219, 2011.
- 47 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.
- 48 Satish B. Rao and Warren D. Smith. Approximating geometrical graphs via “spanners” and “banyans”. In *Proc. 13th ACM Symposium on Theory of Computing (STOC)*, pages 540–550, 1998.
- 49 Liam Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62(3-4):1073–1087, 2012.
- 50 Christian Schindelhauer, Klaus Volbert, and Martin Ziegler. Geometric spanners with applications in wireless networks. *Comput. Geom.*, 36(3):197–214, 2007.
- 51 Michiel H. M. Smid. The well-separated pair decomposition and its applications. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, volume 2. Chapman and Hall/CRC, 2nd edition, 2018.
- 52 Shay Solomon. Euclidean Steiner shallow-light trees. *J. Comput. Geom.*, 6(2):113–139, 2015.
- 53 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k -dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

Distant Representatives for Rectangles in the Plane

Therese Biedl 

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Anna Lubiw 

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Anurag Murty Naredla 

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Peter Dominik Ralbovsky 

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Graeme Stroud 

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

The input to the distant representatives problem is a set of n objects in the plane and the goal is to find a representative point from each object while maximizing the distance between the closest pair of points. When the objects are axis-aligned rectangles, we give polynomial time constant-factor approximation algorithms for the L_1 , L_2 , and L_∞ distance measures. We also prove lower bounds on the approximation factors that can be achieved in polynomial time (unless $P = NP$).

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Distant representatives, blocker shapes, matching, approximation algorithm, APX-hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.17

Supplementary Material Full version appeared at ArXiv.

Full Version: <https://arxiv.org/abs/2108.07751>

Funding *Therese Biedl:* Supported by NSERC.

Anna Lubiw: Supported by NSERC.

Acknowledgements We thank Jeffrey Shallit for help with continued fractions, and we thank anonymous reviewers for their helpful comments.

1 Introduction

The *distant representatives problem* was first introduced by Fiala et al. [17]. The name is a play-on-words on the term “distinct representatives” from Philip Hall’s classic work on bipartite matching [21]. The input is a set of geometric objects in a metric space. The goal is to choose one “representative” point in each object such that the points are distant from each other – more precisely, the objective is to maximize the distance between the closest pair of representative points. In the decision version of the problem, we are given a bound δ and the question is whether we can choose one representative point in each object such that the distance between any two points is at least δ .

The distant representatives problem has applications to map labelling and data visualization. To attach a label to each object, we can find representative points that are at least distance δ apart, and label each object with a ball of diameter δ (a square in L_∞) centred at its representative point.

The distant representatives problem is closely related to dispersion and packing problems. When all the objects are copies of a single object, the distant representatives problem becomes the dispersion problem: to choose k points in a region R to maximize the minimum distance between any two chosen points [3]. Equivalently, the problem is to pack k disjoint discs



© Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 17; pp. 17:1–17:18

Leibniz International Proceedings in Informatics



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

(in the chosen metric) of diameter δ into an expanded region and maximize δ . The distant representatives problem is also related to problems of “imprecise points” where standard computational geometry problems are solved when each input point is only known to lie within some small region [28].

There is a polynomial time algorithm for the distant representatives problem when the objects are segments on a line [32]. This result comes from the scheduling literature – each representative point is regarded as the centre-point of a unit length job. However, as shown by Fiala et al. [17], the decision version of distant representatives becomes NP-hard in 2D when the objects are unit discs for the L_2 norm or unit squares for the L_∞ norm.

Cabello [5] was the first to consider the optimization version of the distant representatives problem. He gave polynomial time approximation algorithms for the cases in 2D where the objects are squares under the L_∞ norm, or discs under the L_2 norm. The squares/discs may intersect and may have different sizes. His algorithms achieve an approximation factor of 2 in L_∞ and $\frac{8}{3}$ in L_2 , with an improvement to 2.24 if the input discs are disjoint. A main idea in his solution is an “approximate-placement” algorithm that chooses representative points from a fine-enough grid using a matching algorithm; small squares/discs that do not contain grid points are handled separately. Cabello noted that the NP-hardness proof of Fiala et al. [17] can be modified to prove that there is no polynomial time approximation scheme (PTAS) for these problems unless $P=NP$. However, no one has given exact lower bounds on the approximation factors that can be achieved in polynomial time.

Our Results

We consider the distant representatives problem for axis-parallel rectangles in the plane. Rectangles are more versatile than squares or circles in many applications, e.g., for labelling rectangular Euler or Venn diagrams [29].

We give polynomial time approximation algorithms to find representative points for the rectangles such that the distance between any two representative points is at least $1/f$ times the optimum. The approximation factors f are given in Table 1 for the L_1 , L_2 , and L_∞ norms. Since rectangles are not fat objects [8], Cabello’s approach of discretizing the problem by choosing representative points from a grid does not extend. Instead, we introduce a new technique of “imprecise discretization” and choose representative points from 1-dimensional shapes (e.g., $+$ -shapes) arranged in a grid. After that, our plan is similar to Cabello’s. First we solve an approximation version of the decision problem – to find representative points so long as the given distance δ is not too large compared to the optimum δ^* . Then we perform a search to find an approximation to δ^* . Unlike previous algorithms which use the real-RAM model, we use the word-RAM model, and thus must address bit complexity issues.

We accompany these positive results with lower bounds on the approximation factors that can be achieved in polynomial time (assuming $P \neq NP$). The lower bounds are shown in Table 1. They apply even in the special case of horizontal and vertical line segments in the plane. The results are proved via gap-producing reductions from Monotone Rectilinear Planar 3-SAT [10]. These are the first explicit lower bounds on approximation factors for the distant representatives problem for any type of object.

Finally, we consider the even more special case of unit-length horizontal line segments, and the decision version of distant representatives. This is even closer to the tractable case of line segments on a line. However, Roeloffzen in his Master’s thesis [30] proved NP-hardness for the L_2 norm. We give a more careful proof that takes care of bit complexity issues, and we show that the problem is NP-complete in the L_1 and L_∞ norms.

■ **Table 1** Bounds on polynomial time approximation factors for the distant representatives problem for axis-aligned rectangles in the plane. A lower bound of x means that an approximation factor less than x implies $P = NP$. (For other L_p norms, there are some constant factors, but we have not optimized them.)

	L_1	L_2	L_∞
upper bound	5	$\sqrt{34} \approx 5.83$	6
lower bound	1.5	1.4425	1.5

For our algorithms and our hardness results, we must deal with bit complexity issues. For rectangles under the L_1 and L_∞ norms, we show that both the optimum value δ^* and the coordinates of an optimum solution have polynomially-bounded bit complexity. In particular, the decision problems lie in NP. The L_2 norm remains more of a mystery, and the decision problem can only be placed in $\exists\mathbb{R}$ (for an explanation of this class, see [6]).

Background

In one dimension, the decision version of the distant representatives problem for intervals on a line was solved by Barbara Simons [32], as a scheduling problem of placing disjoint unit jobs in given intervals. To transform the decision version of distant representatives to the scheduling problem, scale so $\delta = 1$, then expand each interval by $1/2$ on each side. The midpoints of the unit jobs provide the desired solution. Simons's decision algorithm was speeded up to $O(n \log n)$ by Garey et al. [20]. The optimum δ^* can be found using a binary search – in fact there is a discrete set of $O(n^3)$ possible δ^* values, which provides an $O(n^3 \log n)$ algorithm. (We see how to improve this to $O(n^2 \log n)$ but we are not aware of any published improvement.) There has been recent work on the online version of the problem [9]. The (offline) problem is easier when the intervals are disjoint. More generally, the problem is easier when the ordering of the representative points is specified, or is determined – for example if no interval is contained in another then there is an optimum solution where the ordering of the representative points is the same as the ordering of the interval's left endpoints. This “dispersion problem for ordered intervals” can be solved in linear time [26]. In a companion paper to this one, we improved this to a simpler algorithm using shortest paths in a polygon that solves the harder problem of finding the lexicographic maximum list of distances between successive pairs [4].

Cabello [5] gave polynomial time approximation algorithms for the distant representatives problem for balls in the plane, specifically for squares in L_∞ and for discs in L_2 , with approximation factors of 2 and $\frac{8}{3} = 2.6\bar{6}$, respectively. For disjoint discs in L_2 he improved the approximation factor to 2.24. Dumitrescu and Jiang [14] further improved the approximation factor for disjoint discs to 1.414 ($= 1/.707$) by adding LP-based techniques to Cabello's approach. They also considered the case of unit discs, where they gave an algorithm with approximation factor 2.14 ($= 1/.4674$). For disjoint unit discs they gave a very simple algorithm with approximation factor 1.96 ($= 1/.511$). In a follow-up paper Dumitrescu and Jiang [15] gave bounds on the optimum δ^* for balls and cubes in L_2 depending on the minimum area of the union of subsets of k objects – these results have the flavour of Hall's classic condition for the existence of a set of distinct representatives.

The geometric dispersion problem (when all objects are copies of one object) was studied by Bauer and Fekete [3]. They considered the problem of placing k points in a rectilinear polygon with holes to maximize the min L_∞ distance between any two points or between a point and the boundary of the region. Equivalently, the problem is to pack k as-large-as-possible

identical squares into the region. They gave a polynomial time $3/2$ -approximation algorithm, and proved that $14/13$ is a lower bound on the approximation factor achievable in polynomial time. By contrast, if the goal is to pack as many squares of a given size into a region, the famous shifting-grid strategy of Hochbaum and Maas [23] provides a PTAS. Bauer and Fekete use this PTAS to design an approximate decision algorithm for their problem.

It is NP-hard to decide whether a square can be packed with given (different sized) squares [25] or discs [11]. For algorithmic approaches, see the survey [22]. There is a vast literature on the densest packing of equal discs/squares in a region (e.g. a large circle or square) – see the book [33].

Many geometric packing problems suffer from issues of bit complexity. In particular, there are many packing problems that are not known to lie in NP (e.g., packing discs in a square [11]). This issue is addressed in a recent general approach to geometric approximation [16]. Another direction is to prove that packing problems are complete for the larger class $\exists\mathbb{R}$ (existential theory of the reals) [1].

The distant representatives problem is closely related to problems on imprecise points, where each point is only known to lie within some ε -ball, and the worst-case or best-case representative points, under various measures, are considered. Many geometric problems on points (e.g., convex hulls, spanning trees) have been explored under the model of imprecise points [7, 13, 27, 28].

As mentioned above, the distant representatives problem has application to labelling and visualization, specifically it provides a new approach to the problem of labelling (overlapping) rectangular regions or line segments. Most map labelling research is about labelling point features with rectangular labels of a given size, and the objective is to label as many of the points as possible [18]. There is a small body of literature on labelling line features [12, 36], and even less on labelling regions, except by assuming a finite pre-specified set of label positions [34].

Definitions and Preliminaries

Suppose we are given a set \mathcal{R} of n axis-aligned rectangles in 2D. In the *distant representatives problem*, the goal is to choose a point $p(R) \in R$ for each rectangle R in \mathcal{R} so as to maximize the minimum pairwise distance between points, i.e., we want to maximize $\min_{R, R' \in \mathcal{R}} d_\ell(p(R), p(R'))$, where d_ℓ is the distance-function of our choice. We consider here $\ell = 1, 2, \infty$, i.e., the L_1 -distance, the Euclidean L_2 -distance and the L_∞ -distance. We write δ_ℓ^* for the maximum such distance for $\ell \in \{1, 2, \infty\}$, and omit “ ℓ ” when it is clear from the context.

In the *decision version* of the distant representatives problem, we are given not only the rectangles but also a value δ , and we ask whether there exists a set of representative points that have pairwise distances at least δ .

2 Approximating the decision problem

In this section we give an algorithm that takes as input a set \mathcal{R} of axis-aligned rectangles, and a value δ and finds a set of representative points of distance at least δ apart so long as δ is at most some fraction of the optimum, δ^* , for this instance. Let $n = |\mathcal{R}|$ and suppose that the coordinates of the rectangle corners are even integers in the range $[0, D]$ (which guarantees that the rectangle centres also have integer coordinates).

The idea of the algorithm is to overlay a grid of *blocker-shapes* on top of the rectangles as shown in Figure 1, while ensuring that any two blocker-shapes are distance at least δ apart. The hope is to use a matching algorithm to match every rectangle to a unique intersecting

blocker-shape. Then, if rectangle R is matched to blocker-shape B , we choose any point in $R \cap B$ as the representative point for B , which guarantees distance at least δ between representative points since the blocker-shapes are distance at least δ apart. The flaw in this plan is that there may be *small* rectangles that do not intersect a blocker shape. To remedy this, we represent a small rectangle by its centre point, and we eliminate any nearby blocker-shapes before running the matching algorithm.

For the L_1 and L_∞ norms we assume that δ is given as a rational number with at most t digits in the numerator and denominator. Because we are using the word-RAM model where we cannot compute square roots, we will work with δ^2 for the L_2 norm. Thus, for the L_2 norm, we assume that we are given δ^2 as a rational number with at most t digits in the numerator and denominator. The bit size of the input is $\Theta(n \log D + t)$. Similarly, for L_2 , any output representative point (x, y) will be given as (x^2, y^2) . With these nuances of input and output, we express the main result of this section as follows:

► **Theorem 1.** *There exists an algorithm $\text{PLACEMENT}(\delta)$ that, given input $\ell \in \{1, 2, \infty\}$, rectangles \mathcal{R} , and $\delta > 0$, either finds an assignment of representative points for \mathcal{R} of L_ℓ -distance at least δ , or determines that $\delta > \delta_\ell^*/f_\ell$. Here $f_1 = 5$, $f_2 = \sqrt{34} \approx 5.83$, $f_\infty = 6$.*

The run-time of the algorithm is $O(n^2 \log n)$ in the word RAM model, i.e., assuming we can do basic arithmetic on numbers of size $O(\log D + t)$ in constant time.

To describe our algorithm, we think of overlaying the $D \times D$ bounding box of the rectangles with a grid of horizontal and vertical lines such that the diagonal distance across a square of the grid is δ . This means that grid lines are spaced γ_ℓ apart, where $\gamma_1 = \delta/2$, $\gamma_2 = \delta/\sqrt{2}$, and $\gamma_\infty = \delta$. For L_2 we will work with $\gamma_2^2 = \delta^2/2$ which is rational. Note that the algorithm does not explicitly construct the grid. Number the grid lines from left to right and bottom to top, and identify a grid point by its two indices. Note that the number of indices is D/γ_ℓ , so the size of each index is $O(\log D + t)$. We imagine filling the grid with *blocker-shapes*, where the chosen shape depends on the norm L_ℓ that is used – see Figure 1.

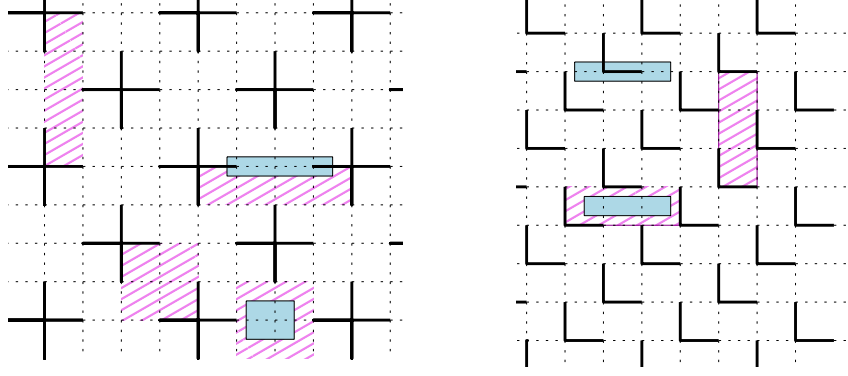
- For $\ell = 1, 2$, we use *+shapes*. Each *+*-shape consists of the four incident grid-segments of one *anchor* grid-point, where (i, j) is the anchor of a *+*-shape iff i is even and $i \equiv j \pmod{4}$.
- For $\ell = \infty$, we use *L-shapes*. Each *L*-shape consists of the two incident grid-segments above and to the right of one *anchor* grid-point, where (i, j) is the anchor of an *L*-shape iff $i \equiv j \pmod{3}$.

Observe that, by our choice of grid size γ_ℓ , any two blocker shapes are distance δ or more apart in the relevant norm.

Algorithm $\text{PLACEMENT}(\delta)$

We now give the rough outline of our algorithm to compute a representative point $p(R)$ for each rectangle R . The details of how to implement each step are given later on. Our algorithm consists of the following steps:

1. Partition the input rectangles into *small* and *big* rectangles. Roughly speaking, a rectangle is *big* if it intersects a blocker-shape, but we give a more precise definition below to deal with intersections on the boundary of the rectangle.
2. For any small rectangle r , let $p(r)$ be the centre of r , i.e., the point where the two diagonals of r intersect each other.
3. If two points $p(r), p(r')$ of two small rectangles r, r' have L_ℓ -distance less than δ , then declare that $\delta^* < f_\ell \delta$, and halt.



■ **Figure 1** Grids and blocker-shapes. We indicate in each a big and a small rectangle (shaded blue) and some cavities (hatched pink). Small rectangles are contained in cavities. (Left) The grid of $+$ -shapes for the L_1 and L_2 norms, with two long cavities and two square cavities. (Right) The grid of L -shapes for the L_∞ -norm with two long cavities.

4. Find all the blocker-shapes that are *owned* by small rectangles, where a blocker-shape B is *owned* by a small rectangle r if $p(r)$ has distance strictly less than δ to some point of B . For L_2 we will enlarge ownership as follows: B is *owned* by r if $d_1(p(r), B) < \sqrt{2}\delta$. To justify that this enlarges ownership, note that $d_1(p, q) \leq \sqrt{2}d_2(p, q)$ so $d_2(p(r), B) < \delta$ implies $d_1(p(r), B) < \sqrt{2}\delta$.
5. Define a bipartite graph H as follows. On one side, H has a vertex for each big rectangle, and on the other side, it has a vertex for each blocker-shape that is not owned by a small rectangle. Add an edge whenever the rectangle intersects the blocker-shape.
6. Construct a subgraph H^- of H as follows. For any big rectangle, if it has degree more than n in H , then arbitrarily delete incident edges until it has degree n . Also delete any blocker-shape that has no incident edges.
7. Compute a maximum matching M in H^- . We say that it *covers all big rectangles* if every big rectangle has an incident matching-edge in M .
8. If M does not cover all big rectangles, then declare that $\delta^* < f_\ell \delta$ and halt.
9. For each big rectangle R let B be the blocker-shape that R is matched to, and let $p(R)$ be an arbitrary point in $B \cap R$. (This exists since (B, R) was an edge.)
10. Return the set $\{p(R)\}$ (for both big and small rectangles R) as an approximate set of distant representatives.

We now define *big* rectangles more precisely. The intuition is that a rectangle is big if it intersects a blocker shape even if δ is decreased by an infinitesimal amount. Note that, as δ decreases, the blocker-shapes change position and size continuously. More formally, a rectangle is *big* with respect to δ if there is some $\varepsilon_0 > 0$ such that for all ε , $0 \leq \varepsilon < \varepsilon_0$, there is a point in the (closed) rectangle and in a blocker-shape (for the blocker-shapes at $\delta - \varepsilon$). The reason for this definition is so the set of big rectangles remains the same if δ is decreased by an infinitesimal amount, a property that becomes relevant when we use the PLACEMENT algorithm to approximately solve the optimization version of distant representatives.

For implementation details and the correctness proof, we need one more definition. A *cavity* is a closed maximal axis-aligned rectangular region with no points of blocker shapes in its interior. We distinguish a *square cavity*, which is a 2×2 block of grid squares (only possible for $+$ -shapes), and a *long cavity* which lies between two consecutive grid lines. For $+$ -shapes a long cavity is a 1×4 or 4×1 block of grid squares, and for L -shapes, a long cavity is a 1×3 or 3×1 block of grid squares. Observe that any small rectangle is contained in a cavity.

Implementation and Runtime. In order to implement the algorithm efficiently we discuss:

- How to test whether a rectangle is big/small.
- How to find the blocker shapes owned by a small rectangle.
- How to construct H^- .
- How to efficiently compute the matching.

We first show how to find which grid square contains a given point. Identify a grid square by the indices of its lower left grid point. Given a point (x, y) in the plane (e.g., a corner of an input rectangle) the vertical grid line just before x has index i where $i\gamma_\ell \leq x < (i+1)\gamma_\ell$ so $i = \lfloor x/\gamma_\ell \rfloor$. For $\ell = 1$, $i = \lfloor 2x/\delta \rfloor$. For $\ell = \infty$, $i = \lfloor x/\delta \rfloor$. For $\ell = 2$, i is the largest natural number such that $i^2 \leq 2x^2/\delta^2$, i.e., i is the integer square root of $\lfloor 2x^2/\delta^2 \rfloor$. The integer square root of a number with $O(\log D + t)$ bits can be found in time $O(\log D + t)$ on a word RAM.

We apply the above procedure $O(n)$ times to find the grid squares of all the rectangles' corners and centres. Using the differences and parities of the indices of the grid squares containing the corners, we can test if a rectangle contains points of blocker shapes in its interior or on its boundary. From this, we can test if a rectangle is big or small in constant time. (Note that our complicated rule is really just testing boundary conditions.)

Each small rectangle r owns a constant number of blocker shapes and these can be found by testing a constant number of grid squares that are near $p(r)$.

Next we show how to construct the bipartite graph H^- and compute a maximum matching. Note that blocker-shapes, which form one vertex set of H^- , are specified using $O(\log D + t)$ bits each, although we do not write that in our run-time bounds. To construct H^- we first build a dictionary for the $O(n)$ blocker-shapes owned by small rectangles. Then for each big rectangle R , enumerate blocker-shapes intersecting R in arbitrary order until we have found n that are not owned by a small rectangle, or until we have found all of them, whichever happens first. The run-time for this step is $O(n^2 \log n)$ which will in fact be the bottleneck in our runtime. The graph H^- has $O(n^2)$ vertices and edges.

To find the maximum matching in H^- , we can use the standard algorithm by Hopcroft and Karp [24] which has run-time $O(\sqrt{\nu}|E|)$, where ν is the size of the maximum matching [31, Theorem 16.5]. We have $\nu \leq n$ and $|E| = O(n^2)$, so the run-time to find the matching is $O(n^{2.5})$. With appropriate further data structures the runtime of computing the matching can be reduced to $O(n\sqrt{n} \log n)$; see the full version. Therefore the runtime becomes $O(n^2 \log n)$.

Correctness

The algorithm outputs either a set of points or a declaration that $\delta_\ell^* < f_\ell \delta$. We first show that the algorithm is correct if it outputs a set of points.

► **Lemma 2.** *If the algorithm returns a point-set, then the L_ℓ -distance between any two points chosen by the algorithm is at least δ .*

Proof. For two small rectangles r, r' , this holds since we test $d_\ell(p(r), p(r'))$ explicitly. For any two big rectangles R, R' , the two assigned points $p(R)$ and $p(R')$ lie on different blocker-shapes, and hence have distance at least δ . For any big rectangle R and small rectangle r , point $p(R)$ lies on a blocker-shape that is not owned by r , so the blocker shape, and hence $p(R)$, has distance at least δ from $p(r)$. ◀

If the algorithm does not output a set of points, then it outputs a declaration that δ is too large compared to the optimum δ^* , viz., $\delta_\ell^* < f_\ell \delta$. This declaration is made either in Step 3 because the points chosen for small rectangles are too close, or in Step 8 because no matching

is found. We must prove correctness in each case, Lemma 3 for Step 3, and Lemma 4 for Step 8. In the remainder of this section we let $p^*(R)$, $R \in \mathcal{R}$ denote an optimum set of distant representatives, i.e., $p^*(R)$ is a point in R and every two such points have L_ℓ -distance at least δ_ℓ^* .

► **Lemma 3.** *If two points $p(r), p(r')$ of two small rectangles r, r' have distance less than δ , then $\delta_\ell^* < f_\ell \delta$.*

Proof. We first show that for any small rectangle r , points $p^*(r)$ and $p(r)$ are close together, specifically, $d_\ell(p^*(r), p(r)) \leq 2.5\gamma_\ell$. Because L_1 -distance dominates L_2 and L_∞ -distances, it suffices to prove that $d_1(p^*(r), p(r)) \leq 2.5\gamma_\ell$. Any small rectangle is contained in a cavity. The L_1 diameter of a cavity (i.e., the maximum distance between any two points in the cavity) is at most $5\gamma_\ell$ – it is $5\gamma_\ell$ for a long cavity with $+$ -shapes; $4\gamma_\ell$ for a square cavity with $+$ -shapes; and $4\gamma_\ell$ for a long cavity with L -shapes. This implies that any point of r is within distance $2.5\gamma_\ell$ from $p(r)$, the centre of rectangle r .

Now consider two small rectangles r and r' with $d_\ell(p(r), p(r')) < \delta$. We will bound the distance between $p^*(r)$ and $p^*(r')$ by applying the triangle inequality:

$$d_\ell(p^*(r), p^*(r')) \leq d_\ell(p^*(r), p(r)) + d_\ell(p(r), p(r')) + d_\ell(p(r'), p^*(r')) < 2.5\gamma_\ell + \delta + 2.5\gamma_\ell = \delta + 5\gamma_\ell.$$

Plugging in the values $\gamma_1 = \delta/2$, $\gamma_2 = \delta/\sqrt{2}$, and $\gamma_\infty = \delta$, we obtain bounds of 3.5δ , $(1 + 5/\sqrt{2})\delta \approx 4.5\delta$, and 6δ , respectively. Since $f_1 = 5$, $f_2 \approx 5.8$, and $f_\infty = 6$, these bounds are at most $f_\ell \delta$ in all three cases. Thus $\delta^* < f_\ell \delta$, as required. ◀

► **Lemma 4.** *If there is no matching M in H^- that covers all big rectangles, then $\delta_\ell^* < f_\ell \delta$.*

Proof. We prove the contrapositive, using the following plan. Take an optimal set of distant representatives, $p^*(R)$, $R \in \mathcal{R}$ with L_ℓ -distance $\delta_\ell^* \geq f_\ell \delta$. For any big rectangle R , we “round” $p^*(R)$ to a point $b(R)$ that is in R and on a blocker-shape $B(R)$. More precisely, we define $b(R)$ to be a point that is in R , on a blocker-shape, and closest (in L_ℓ distance) to $p^*(R)$. In case of ties, choose $b(R)$ so that the smallest rectangle containing $p^*(R)$ and $b(R)$ is minimal (this is only relevant in L_∞). Break further ties arbitrarily. Observe that $b(R)$ exists, since a big rectangle contains blocker-shape points. Define $B(R)$ to be the blocker-shape containing $b(R)$.

By Lemma 5 (stated below) the pairs $R, B(R)$ form a matching in H that covers all big rectangles. We convert this to a matching in H^- by repeatedly applying the following exchange step. If big rectangle R is matched to a blocker shape $B(R)$ that is not in H^- , then R has degree exactly n in H^- . Not all its n neighbours can be used in the current matching since there are at most $n - 1$ big rectangles other than R . So change the matching-edge at R to go to one of the unmatched neighbours in H^- instead. ◀

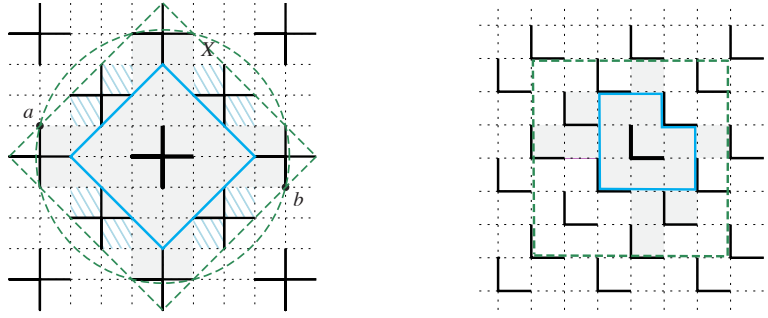
► **Lemma 5.** *Let R be a big rectangle and let $B = B(R)$. If $\delta^* \geq f_\ell \delta$ then (1) no other big rectangle R' has $B(R') = B$, and (2) no small rectangle owns B .*

The general idea to prove this lemma is to show that either type of collision ($B(R) = B(R')$ or $B(R)$ owned by r) gives points p^* that are “close together”, where close together means in a ball of appropriate radius centred at the anchor of $B(R)$.

Let $C_\ell(B)$ be the open L_ℓ -ball centred at the anchor of B and with diameter $f_\ell \delta$. See Figure 2 and note that the diameter of the ball in the appropriate metric is:

$$f_1 \delta = 5\delta = 10(\delta/2) = 10\gamma_1 \quad f_2 \delta = \sqrt{34}\delta = \sqrt{68}(\delta/\sqrt{2}) = \sqrt{68}\gamma_2 \quad f_\infty \delta = 6\delta = 6\gamma_\infty$$

We need a few claims localizing $p^*(R)$ relative to $b(R)$:



■ **Figure 2** A blocker shape B (heavy black) and $C_\ell(B)$, the L_ℓ -ball of diameter $f_\ell\delta$ centred at B 's anchor (dashed green) which is a diamond for L_1 (left), a circle for L_2 (left), and a square for L_∞ (right). The long cavities that touch B are shaded gray. If a small rectangle r owns B , then $p(r)$ lies in C' (in cyan), and r is contained in the union of the gray and blue-hatched regions.

▷ **Claim 6.** For any big rectangle R , the points $b(R)$ and $p^*(R)$ lie in one long cavity.

Proof. Let T be the rectangle with corners $p^*(R)$ and $b(R)$. By definition of $b(R)$, there are no points of blocker-shapes in or on the boundary of T except $b(R)$. Thus T is contained in a cavity. Furthermore, if T is contained in a square cavity, then we claim that T does not contain the central grid point of the square cavity in its interior (otherwise $b(R)$ could not be the unique point of a blocker shape in T , see Figure 1). Thus T is contained in a long cavity. ◁

▷ **Claim 7.** Let B be a blocker shape. Let R be a big rectangle with $B(R) = B$. Then $p^*(R)$ is contained in the ball $C_\ell(B)$.

Proof. By the previous lemma, $p^*(R)$ lies in a long cavity that contains a point of B . From Figure 2 we see that any long cavity that contains a point of B lies inside the closed ball $C_\ell(B)$. Furthermore, note that if $p^*(R)$ lies on the boundary of the ball, then it lies on a different blocker shape, contrary to $B(R) = B$. Thus $p^*(R)$ is contained in $C_\ell(B)$. ◁

▷ **Claim 8.** Let B be a blocker shape. Let r be a small rectangle that owns B . Then $p^*(r)$ is contained in the ball $C_\ell(B)$.

Proof. Recall that $p(r)$ is the centre of r , and that, by the definition of ownership, for $\ell = 1, \infty$ we have $d_\ell(p(r), B) < \delta$, and for $\ell = 2$ we have $d_1(p(r), B) < \sqrt{2}\delta$. Such points $p(r)$ lie in the open region C' drawn in cyan in Figure 2. Here C' is the Minkowski sum of an open ball with B where we use an L_1 -ball of radius δ for $\ell = 1$, an L_1 -ball of radius $\sqrt{2}\delta$ for $\ell = 2$ and an L_∞ -ball of radius δ for $\ell = \infty$.

We next show that r is contained in the ball $C_\ell(B)$. For $\ell = \infty$, r must lie in a long cavity intersecting C' , i.e. in the open shaded gray region, thus in $C_\infty(B)$.

For $\ell = 1, 2$, r is contained in a cavity that intersects C' . The union of the cavities that intersect C' consists of the gray and blue-hatched region plus the grid square X and its symmetric counterparts. But observe that a small rectangle that contains points of X has a centre outside C' . Therefore r is contained in $C_\ell(B)$. ◁

Proof of Lemma 5. Let R be a big rectangle and let $B = B(R)$. By Claim 7, $p^*(R)$ lies in $C_\ell(B)$. If there is another big rectangle R' with $B(R') = B$, then $p^*(R')$ also lies in $C_\ell(B)$. If there is a small rectangle r that owns B , then by Claim 8, $p^*(r)$ lies in $C_\ell(B)$.

In either case, the distance between $p^*(R)$ and the other p^* point is less than $f_\ell\delta$, since that is the diameter of $C_\ell(B)$. Thus $\delta^* < f_\ell\delta$. ◀

3 Approximating the optimization problem

In this section we use PLACEMENT to design approximation algorithms for the optimization version of the distant representatives problem for rectangles:

► **Theorem 9.** *There is an f_ℓ -approximation algorithm for the distant representatives problem on rectangles in the L_ℓ -norm, $\ell = 1, 2, \infty$ with run time $O(n^2(\log n)^2)$ for L_∞ and run time $O(n^2 \text{polylog}(nD))$ for L_1 and L_2 . Here (as before) $f_1 = 5, f_2 = \sqrt{34} \approx 5.83, f_\infty = 6$.*

One complicating factor is that PLACEMENT is not monotone, i.e., it may happen that PLACEMENT fails for a value δ but succeeds for a larger value δ' . We note that Cabello's algorithm [5] behaves the same way. We deal with L_∞ in Section 3.1 and with L_1 and L_2 in Section 3.2.

We need some upper and lower bounds on δ^* . Note that if the input contains two rectangles that are single identical points, then $\delta^* = 0$. Since this is easily detected, we assume from now on that no two input rectangles are single identical points.

▷ **Claim 10.** We have $1/n \leq \delta^* \leq 2D$.

Proof. The upper bound is obvious. For the lower bound, place a grid of points distance $\frac{1}{n}$ apart. All rectangles with non-zero dimensions will intersect at least $n + 1$ points, and single-point rectangles will hit one point. Since no two single-point rectangles are identical, they can be matched to the sole grid point that overlaps the rectangle. The remaining rectangles can be matched trivially. ◁

Note that a solution with distance at least $\frac{1}{n}$ can be found easily, following the steps above.

3.1 Optimization problem for L_∞

For the L_∞ norm we use the following result about the possible optimum values; a proof is in the full version.

► **Lemma 11.** *In L_∞ , δ_∞^* takes on one of the $O(n^3)$ values of the form $(t - b)/k$ where $k \in \{1, \dots, n\}$ and t, b are rectangle coordinates.*

Let Δ be the set of $O(n^3)$ values from the lemma. We can compute the set Δ in $O(n^3)$ time and sort it in $O(n^3 \log n)$ time. Say $\Delta = \{d_1, d_2, \dots, d_N\}$ in sorted order, and set $c_i := d_i / f_\infty$. Because of non-monotonicity, we cannot efficiently find the maximum c_i for which PLACEMENT succeeds. Instead, we use binary search to find i such that $\text{PLACEMENT}(c_i)$ succeeds but $\text{PLACEMENT}(c_{i+1})$ fails. Therefore $\delta_\infty^* < f_\infty c_{i+1} = d_{i+1}$ which implies that $\delta_\infty^* \leq d_i = f_\infty c_i$ and the representative points found by $\text{PLACEMENT}(c_i)$ provide an f_∞ -approximation.

To initialize the binary search, we first run $\text{PLACEMENT}(c_N)$, and, if it succeeds, return its computed representative points since they provide an f_∞ -approximation of the optimum assignment. Also note that $\text{PLACEMENT}(c_1)$ must succeed – if it fails then $\delta_\infty^* < f_\infty c_1 = d_1$, which contradicts $\delta_\infty^* \in \Delta$. Thus we begin with the interval $[1, N]$ and perform binary search on within this interval to find a value i such that $\text{PLACEMENT}(c_i)$ succeeds but $\text{PLACEMENT}(c_{i+1})$ fails.

We can get away with sorting just the $O(n^2)$ numerators and performing an implicit binary search, to avoid the cost of generating and sorting all of Δ . Let t be the sorted array of $O(n^2)$ numerators, which takes $O(n^2 \log n)$ time to generate and sort. The denominators are just $[n]$, so there is no need to generate and sort it explicitly. Define the implicit *sorted*

matrix a , where $a[r, c] = t[r]/(n - c)$, for $0 \leq r \leq |t| - 1, 0 \leq c \leq n - 1$. Each entry of a can be computed in $O(1)$ time. Since the matrix is sorted, the matrix selection algorithm of Frederickson and Johnson [19] can be used to get an element of Δ at the requested index in $O(n \log n)$ time. Using selection, one can perform the binary search on Δ implicitly. While accessing elements of Δ takes more time, it is still less than the time to call `PLACEMENT` on the element accessed. Each iteration of the binary search is dominated by the runtime of `PLACEMENT`, so the total runtime is $O(n^2(\log n)^2)$. This proves Theorem 9 for L_∞ .

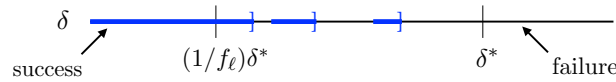
3.2 Optimization problem for L_1 and L_2

In this section we give an approximation algorithm for the optimization version of distant representatives for rectangles in the L_1 and L_2 norms. Define a *critical* value to be a right endpoint

We

► **Len**

success



■ **Figure 3** An illustration of critical values.

Thus our problem reduces to finding a critical value.

► **Lemma 13.** *The `PLACEMENT` algorithm can be modified to detect critical values.*

► **Lemma 14.** *In L_1 any critical value δ is a rational number with numerator and denominator at most $4Dn$. In L_2 for any critical value δ , δ^2 is a rational number with numerator and denominator at most $8D^2n^2$.*

Based on these lemmas, we use continued fractions to find a critical value. We need the following properties of continued fractions.

1. A continued fraction has the form $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots + \frac{1}{a_k}}}$, where the a_i 's are natural numbers.
2. Every positive rational number $\frac{a}{b}$ has a continued fraction representation. Furthermore, the number of terms, k , is $O(\log(\max\{a, b\}))$. This follows from the fact that computing the continued fraction representation of $\frac{a}{b}$ exactly parallels the Euclidean algorithm; see [2, Theorem 4.5.2] or the wikipedia page on the Euclidean algorithm [35]. For the same reason, each a_i is bounded by $\max\{a, b\}$.
3. Suppose we don't know $\frac{a}{b}$ explicitly, but we know some bound G such that $a, b \leq G$, and we have a test of whether a partial continued fraction is greater than, less than, or equal to $\frac{a}{b}$. Then we can find the continued fraction representation of $\frac{a}{b}$ as follows. For $i = 1 \dots \log G$, use binary search on $[2..G]$ to find the best value for a_i . Note that the continued fraction with i terms is increasing in a_i for even i and decreasing for odd i , and we adjust the binary search correspondingly. In each step, we have a lower bound and an upper bound on $\frac{a}{b}$ and the step shrinks the interval. If the test runs in time T , then the time to find the continued fraction for $\frac{a}{b}$ is $O(T(\log G)^2)$, plus the cost of doing arithmetic on continued fractions (with no T factor).

Algorithm for L_1, L_2 . Run the continued fraction algorithm using PLACEMENT (enhanced to detect a critical value) as the test. The only difference from the above description is that we do not have a specific target $\frac{a}{b}$; rather, our interval contains at least one critical value and we search until we find one. At any point we have two values b_l and b_u both represented as continued fractions, where PLACEMENT succeeds at b_l and fails at b_u , so there is at least one critical value between them. We can use the initial interval $[1/n, 2D]$. To justify this, note that if $\text{PLACEMENT}(\frac{1}{n})$ fails, then $f_\ell/n > \delta^*$ by Theorem 1, so we get an f_ℓ -approximation by using the representative points for $\delta = \frac{1}{n}$ (see the remark after Claim 10).

For the runtime, we use the bound $O(T(\log G)^2)$ from point 3 above, plugging in $T = O(n^2 \log n)$ for PLACEMENT and the bounds on G from Lemma 14, to obtain a runtime of $O(n^2 \text{polylog}(nD))$, which proves Theorem 9 for L_1 and L_2 .

The run-time for Theorem 9 can actually be improved to $O(n^2(\log n)^2)$ (i.e., without the dependence on $\log D$) with an approach that is very specific to the problem at hand (and similar to Cabello's approach). The details are complicated for such a relatively small improvement and hence omitted here.

Missing proofs. For space reasons we can here only give the briefest sketch of the proofs of Lemmas 12, 13, and 14; details are in the full version. A crucial ingredient is to study what must have happened if $\text{PLACEMENT}(\delta)$ goes from success to failure (when viewing its outcome as a function that changes over time δ).

► **Observation 15.** *Assume $\text{PLACEMENT}(\delta)$ succeeds but $\text{PLACEMENT}(\delta')$ fails for some $\delta' > \delta$. Then at least one of the following events occurs as we go from δ to δ' :*

1. *the set of small/big rectangles changes,*
2. *the distance between the centres of two small rectangles equals $\hat{\delta}$ for some $\delta \leq \hat{\delta} < \delta'$,*
3. *the set of blocker-shapes owned by a small rectangle increases,*
4. *the set of blocker-shapes intersecting a big rectangle decreases.*

Roughly speaking, Lemma 12 can now be shown by arguing that such events do not happen in a sufficiently small time-interval before PLACEMENT fails (hence the intervals where it fails are open on the left). Lemma 14 holds because there necessarily must have been an event at time δ , and we can analyze the coordinates when events happen. Finally Lemma 13 is achieved by running PLACEMENT at time δ and also symbolically at time $\delta + \varepsilon$.

4 Hardness results

In this section we outline NP-hardness and APX-hardness results for the distant representatives problem. For complete details see the full version of the paper. We first show that, even for the special case of unit horizontal segments, the decision version of the problem is NP-complete for L_1 and L_∞ and NP-hard for L_2 (where bit complexity issues prevent us from placing the problem in NP). This L_2 result was proved previously by Roeloffzen in his Master's thesis [30, Section 2.3] but we add details regarding bit complexity that were missing from his proof.

Next, we enhance our reductions to “gap-producing reductions” to obtain lower bounds on the approximation factors that can be achieved in polynomial time. Since our goal is to compare with our approximation algorithms for rectangles, we consider the more general case of horizontal and vertical segments in the plane (not just unit horizontals). Our main result is that, assuming $P \neq NP$, no polynomial time approximation algorithm achieves a factor better than 1.5 in L_1 and L_∞ and 1.4425 in L_2 .

Our reductions are from the NP-complete problem Monotone Rectilinear Planar 3-SAT [10] in which each clause has either three positive literals or three negative literals, each variable is represented by a thin vertical rectangle at x -coordinate 0, each positive [negative] clause is represented by a thin vertical rectangle at a positive [negative, resp.] x -coordinate, and there is a horizontal line segment joining any variable to any clause that contains it. See Figure 4(a) for an example instance of the problem. For n variables and m clauses, the representation can be on an $O(m) \times O(n + m)$ grid.

4.1 NP-hardness

► **Theorem 16.** *The decision version of the distant representatives problem for unit horizontal segments in the L_1, L_2 or L_∞ norm is NP-hard.*

Lemma 11 implies that the decision problem lies in NP for the L_∞ norm, even for rectangles. In the full version we show the same for L_1 , and we discuss the bit complexity issues that prevent us from placing the decision problem in NP for the L_2 norm.

For our reduction from Monotone Rectilinear Planar 3-SAT we first modify the representation so that each clause rectangle has fixed height and is connected to its three literals via three “wires” – the middle one remains horizontal, the bottom one bends to enter the clause rectangle from the bottom, and the top one bends twice to enter the clause rectangle from the far side. See Figure 4(b). Each wire is directed from the variable to the clause, and represents a literal. The representation is still on an $O(m) \times O(n + m)$ grid.

To complete the reduction to the distant representatives problem we replace the rectangles with variable and clause gadgets constructed from unit horizontal intervals, and also implement the wires using such intervals. The details, which can be found in the full version of the paper, depend on the norm L_ℓ , $\ell = 1, 2, \infty$. We also set a value of δ_ℓ to obtain a decision problem that asks if there is an assignment of a representative point to each interval that is *valid*, i.e., such that no two points are closer than δ_ℓ . We set $\delta_1 = 2$, $\delta_2 = \frac{13}{5}$, and $\delta_\infty = \frac{1}{2}$. An example of the construction for L_1 (with $\delta_1 = 2$) is shown in Figure 4(c).

For the L_2 norm, the bit complexity issue missed in Roeloffzen’s reduction [30, Section 2.3] is that the interval endpoints and their distances must have polynomially-bounded bit complexity. We resolve this by using Pythagorean triples (see Figure 5(a)).

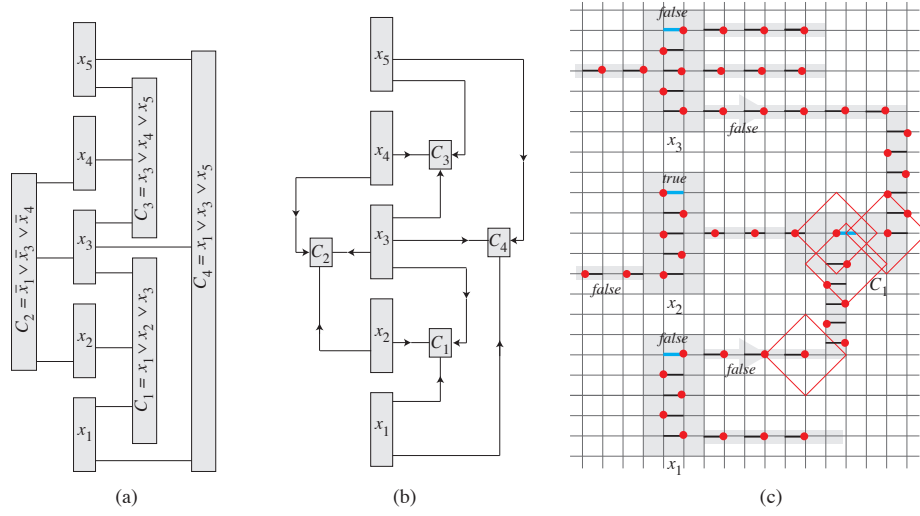
4.2 APX-hardness

In this section, we prove hardness-of-approximation results for the distant representatives problem on horizontal and vertical segments in the plane. Specifically, we prove lower bounds on the approximation factors that can be achieved in polynomial time, assuming $P \neq NP$.

► **Theorem 17.** *For $\ell = 1, 2, \infty$, let g_ℓ be the constant shown in Table 2. Suppose $P \neq NP$. Then, for the L_ℓ norm, there is no polynomial time algorithm with approximation factor less than g_ℓ for the distant representatives problem for horizontal and vertical segments.*

■ **Table 2** Best approximation ratios that can be achieved unless $P=NP$.

	L_1	L_2	L_∞
lower bound	$g_1 = 1.5$	$g_2 = 1.4425$	$g_\infty = 1.5$



■ **Figure 4** (a) An instance of Monotone Rectilinear Planar 3-SAT. (b) The modified representation used for our NP-hardness proofs, with wires from variable to clause gadgets. (c) A detail of our NP-hardness construction for clause $C_1 = x_1 \vee x_2 \vee x_3$ in the L_1 norm showing how the truth-value setting $x_1 = \text{False}$, $x_2 = \text{True}$, $x_3 = \text{False}$, permits representative points (shown as red dots) at distance at least $\delta_1 = 2$.

We prove Theorem 17 using a *gap reduction*. This standard approach is based on the fact that if there were polynomial time approximation algorithms with approximation factors better than g_ℓ then the *gap versions* of the problem (as stated below) would be solvable in polynomial time. Thus, proving that the gap versions are NP-hard implies that there are no polynomial time g_ℓ -approximation algorithms unless $P=NP$.

Recall that δ_ℓ^* is the max over all assignments of representative points, of the min distance between two points.

Gap Distant Representatives Problem.

Input: A set I of horizontal and vertical segments in the plane.

Output:

- YES if $\delta_\ell^*(I) \geq 1$;
- NO if $\delta_\ell^*(I) \leq 1/g_\ell$;
- and it does not matter what the output is for other inputs.

To prove Theorem 17 it therefore suffices to prove:

► **Theorem 18.** *The Gap Distant Representatives problem is NP-hard.*

This is proved via a reduction from Monotone Rectilinear Planar 3-SAT, much like in the previous section. The gadgets are simpler because we can use vertical segments, but we must prove stronger properties. Given an instance Φ of Monotone Rectilinear Planar 3-SAT we construct in polynomial time a set of horizontal and vertical segments I such that:

▷ **Claim 19.** If Φ is satisfiable then $\delta_\ell^*(I) = 1$.

▷ **Claim 20.** If Φ is not satisfiable then $\delta_\ell^*(I) \leq 1/g_\ell$.

Thus a polynomial time algorithm for the Gap Distant Representatives problem yields a polynomial time algorithm for Monotone Rectilinear Planar 3-SAT. We give some of the reduction details, but defer the proofs of the claims to the full version.

Reduction details

We reduce directly from Monotone Rectilinear Planar 3-SAT.

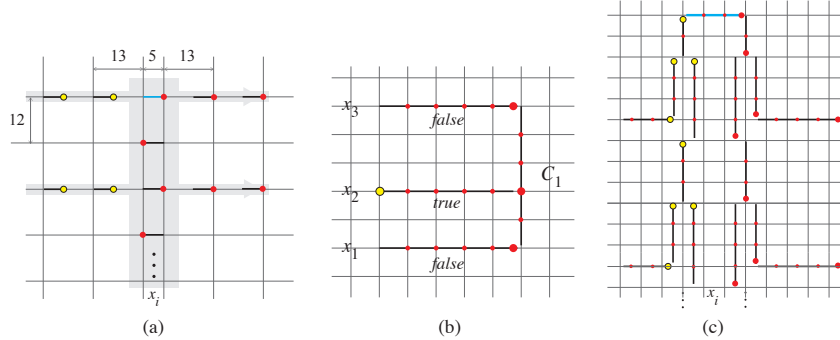


Figure 5 (a) A variable gadget for NP-hardness for L_2 , based on Pythagorean triple 5, 12, 13. To achieve $\delta = 13$ the representative point for the variable interval (in cyan) is forced to the left (true) or the right (false) in which case intervals on the right are also forced. (b) A clause gadget for the APX-hardness reduction, with three horizontal wires attached. For clarity, segments are not drawn all the way to their endpoints. Wires x_1 and x_2 are in the false setting and wire x_3 is in the true setting, which allows the representative point for C_1 to be placed where the x_3 wire meets it, while keeping representative points at least distance 1 apart. (c) The basic splitter gadget for APX-hardness for L_∞ placed on the half grid and showing two wires extending left and two right. The variable segment (in thick cyan) for the variable x_i has its representative point (the large red dot) at the right, which is the false setting. The representative points shown by large red/yellow dots are distance at least 1 apart in L_∞ .

Wire. A wire is a long horizontal segment with 0-length segments at unit distances along it, except at its left and right endpoints. See Figure 5(b). The representative point for a 0-length segment must be the single point in the segment (shown as small red dots in the figure). As before, a wire is directed from the variable gadget to the clause gadget. We distinguish a “false setting” where the wire has its representative point within distance 1 of its forward end (at the clause gadget) and a “true setting” where the wire has its representative point within distance 1 of its tail end (at the variable gadget).

Clause gadget. A clause gadget is a vertical segment. Three wires corresponding to the three literals in the clause meet the vertical segment as shown in Figure 5(b). There are 0-length segments at unit distances along the clause interval except where the three wires meet it.

Variable gadget. A variable segment has length 3, with two 0-length segments placed 1 and 2 units from the endpoints. A representative point in the right half corresponds to a false value for the variable, and a representative point in the left half corresponds to a true value. In order to transmit the variable’s value to all the connecting horizontal wires we build a “splitter” gadget. The basic splitter gadget for L_∞ is shown in Figure 5(c). The same splitter gadget works for the other norms but we can improve the lower bounds using modified splitter gadgets as described in the full version.

5 Conclusions

We gave good approximation algorithms for the distant representatives problem for rectangles in the plane using a new technique of “imprecise discretization” where we limit the choice of representative points not to a discrete set but to a set of one-dimensional “shapes”. This technique may be more widely applicable, and can easily be tailored, for example by using a weighted matching algorithm to prefer representative points near the centres of rectangles.

We also gave the first explicit lower bounds on approximation factors that can be achieved in polynomial time for distant representatives problems.

Besides the obvious questions of improving the approximation factors, the run-times, or the lower bounds, we mention several other avenues for further research.

1. Is the distant representatives problem for rectangles in L_2 hard for existential theory of the reals? Recently, some packing problems have been proved $\exists\mathbb{R}$ -complete [1], but they seem substantially harder.
2. Is there a good [approximation] algorithm for any version of distant representatives for a *lexicographic* objective function. For example, suppose we wish to maximize the smallest distance between points, and, subject to that, maximize the second smallest distance, and so on. Or suppose we ask to lexicographically maximize the sorted vector consisting of the n distances from each chosen point to its nearest neighbour. For the case of *ordered* line segments in 1D there is a linear time algorithm to lexicographically minimize the sorted vector of distances between successive pairs of points [4]. It is an open problem to extend this to unordered line segments.
3. What about weighted versions of distant representatives? Here each rectangle R has a weight $w(R)$, and rather than packing disjoint balls of radius δ we pack disjoint balls of radius $w(R)\delta$ centred at a representative point $p(R)$ in R . Again, there is a solution for ordered line segments in 1D [4].

References

- 1 Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. Framework for $\exists\mathbb{R}$ -completeness of two-dimensional packing problems. *arXiv preprint arXiv:2004.07558*, 2020. URL: <https://arxiv.org/abs/2004.07558>.
- 2 Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory: Efficient Algorithms*, volume 1. MIT press, 1996.
- 3 Christoph Baur and Sándor P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001. doi:10.1007/s00453-001-0022-x.
- 4 Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Dispersion for intervals: A geometric approach. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 37–44. SIAM, 2021. doi:10.1137/1.9781611976496.4.
- 5 Sergio Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007. doi:10.1016/j.jalgor.2004.06.009.
- 6 Jean Cardinal. Computational geometry column 62. *ACM SIGACT News*, 46(4):69–78, 2015. doi:10.1145/2852040.2852053.
- 7 Erin Chambers, Alejandro Erickson, Sándor P. Fekete, Jonathan Lenchner, Jeff Sember, Venkatesh Srinivasan, Ulrike Stege, Svetlana Stolpner, Christophe Weibel, and Sue Whitesides. Connectivity graphs of uncertainty regions. *Algorithmica*, 78(3):990–1019, 2017. doi:10.1007/s00453-016-0191-2.
- 8 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003.

- 9 Jing Chen, Bo Li, and Yingkai Li. Efficient approximations for the online dispersion problem. *SIAM Journal on Computing*, 48(2):373–416, 2019. doi:10.1137/17M1131027.
- 10 Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In *International Computing and Combinatorics Conference*, pages 216–225. Springer, 2010. doi:10.1007/978-3-642-14031-0_25.
- 11 Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. *arXiv preprint arXiv:1008.1224*, 2010. URL: <https://arxiv.org/abs/1008.1224>.
- 12 Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M.E. Moret, and Binhai Zhou. Map labeling and its generalizations. In *Proc. 8th Ann. ACM/SIAM Symp. Discrete Algs.(SODA97)*, pages 148–157. SIAM, 1997.
- 13 Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum-and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, 56(1):220–250, 2015. doi:10.1007/s00224-014-9591-3.
- 14 Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory of Computing Systems*, 51(2):125–142, 2012. doi:10.1007/s00224-011-9331-x.
- 15 Adrian Dumitrescu and Minghui Jiang. Systems of distant representatives in Euclidean space. *Journal of Combinatorial Theory, Series A*, 134:36–50, 2015. doi:10.1016/j.jcta.2015.03.006.
- 16 Jeff Erickson, Ivor van der Hoog, and Tillmann Miltzow. Smoothing the gap between NP and $\exists\mathbb{R}$. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1022–1033. IEEE, 2020.
- 17 Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005. doi:10.1016/j.dam.2004.02.018.
- 18 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 281–288, 1991.
- 19 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.
- 20 Michael R. Garey, David S. Johnson, Barbara B. Simons, and Robert Endre Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981. doi:10.1137/0210018.
- 21 Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.
- 22 Mhand Hifi and Rym M’hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009, 2009. doi:10.1155/2009/150624.
- 23 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. doi:10.1016/j.orl.2010.07.004.
- 24 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.
- 25 Joseph Y.T. Leung, Tommy W. Tam, Chin S. Wong, Gilbert H. Young, and Francis Y.L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990. doi:10.1016/0743-7315(90)90019-L.
- 26 Shimin Li and Haitao Wang. Dispersing points on intervals. *Discrete Applied Mathematics*, 239:106–118, 2018. doi:10.1016/j.dam.2017.12.028.
- 27 Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235, 2010. doi:10.1007/s00453-008-9174-2.
- 28 Maarten Löffler and Marc van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419–433, 2010. doi:10.1016/j.comgeo.2009.03.007.

- 29 Roger J. Marshall. Scaled rectangle diagrams can be used to visualize clinical and epidemiological data. *Journal of Clinical Epidemiology*, 58(10):974–981, 2005. doi:10.1016/j.jclinepi.2005.01.018.
- 30 M.J.M. Roeloffzen. Finding structures on imprecise points. Master’s thesis, TU Eindhoven, 2009. URL: <https://www.win.tue.nl/~mroeloff/papers/thesis-roeloffzen2009.pdf>.
- 31 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer Science & Business Media, 2003.
- 32 Barbara Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science*, pages 246–252. IEEE, 1978. doi:10.1109/SFCS.1978.4.
- 33 Péter Gábor Szabó, Mihaly Csaba Markót, Tibor Csendes, Eckard Specht, Leocadio G. Casado, and Inmaculada García. *New Approaches to Circle Packing in a Square: with Program Codes*, volume 6. Springer Science & Business Media, 2007.
- 34 Frank Wagner, Alexander Wolff, Vikas Kapoor, and Tycho Strijk. Three rules suffice for good label placement. *Algorithmica*, 30(2):334–349, 2001.
- 35 Wikipedia contributors. Euclidean algorithm — Wikipedia, the free encyclopedia, 2021. [Online; accessed 28-June-2021]. URL: https://en.wikipedia.org/w/index.php?title=Euclidean_algorithm&oldid=1027503317.
- 36 Alexander Wolff, Lars Knipping, Marc van Kreveld, Tycho Strijk, and Pankaj K. Agarwal. A simple and efficient algorithm for high-quality line labeling. In Peter Atkinson, editor, *GIS and GeoComputation*. Taylor and Francis, 2000. doi:10.1201/9781482268263.

Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles

Daide Bilò 

Department of Humanities and Social Sciences, University of Sassari, Italy

Sarel Cohen 

Hasso Plattner Institute, University of Potsdam, Germany

Tobias Friedrich 

Hasso Plattner Institute, University of Potsdam, Germany

Martin Schirneck 

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Given a graph with a distinguished source vertex s , the Single Source Replacement Paths (SSRP) problem is to compute and output, for any target vertex t and edge e , the length $d(s, t, e)$ of a shortest path from s to t that avoids a failing edge e . A Single-Source Distance Sensitivity Oracle (Single-Source DSO) is a compact data structure that answers queries of the form (t, e) by returning the distance $d(s, t, e)$. We show how to deterministically compress the output of the SSRP problem on n -vertex, m -edge graphs with integer edge weights in the range $[1, M]$ into a Single-Source DSO that has size $O(M^{1/2}n^{3/2})$ and query time $\tilde{O}(1)$. We prove that the space requirement is optimal (up to the word size). Our techniques can also handle vertex failures within the same bounds.

Chechik and Cohen [SODA 2019] presented a combinatorial, randomized $\tilde{O}(m\sqrt{n} + n^2)$ time SSRP algorithm for undirected and unweighted graphs. We derandomize their algorithm with the same asymptotic running time and apply our compression to obtain a deterministic Single-Source DSO with $\tilde{O}(m\sqrt{n} + n^2)$ preprocessing time, $O(n^{3/2})$ space, and $\tilde{O}(1)$ query time. Our combinatorial Single-Source DSO has near-optimal space, preprocessing and query time for unweighted graphs, improving the preprocessing time by a \sqrt{n} -factor compared to previous results with $o(n^2)$ space.

Grandoni and Vassilevska Williams [FOCS 2012, TALG 2020] gave an algebraic, randomized $\tilde{O}(Mn^\omega)$ time SSRP algorithm for (undirected and directed) graphs with integer edge weights in the range $[1, M]$, where $\omega < 2.373$ is the matrix multiplication exponent. We derandomize it for undirected graphs and apply our compression to obtain an algebraic Single-Source DSO with $\tilde{O}(Mn^\omega)$ preprocessing time, $O(M^{1/2}n^{3/2})$ space, and $\tilde{O}(1)$ query time. This improves the preprocessing time of algebraic Single-Source DSOs by polynomial factors compared to previous $o(n^2)$ -space oracles.

We also present further improvements of our Single-Source DSOs. We show that the query time can be reduced to a constant at the cost of increasing the size of the oracle to $O(M^{1/3}n^{5/3})$ and that all our oracles can be made path-reporting. On sparse graphs with $m = O(\frac{n^{5/4-\varepsilon}}{M^{7/4}})$ edges, for any constant $\varepsilon > 0$, we reduce the preprocessing to randomized $\tilde{O}(M^{7/8}m^{1/2}n^{11/8}) = O(n^{2-\varepsilon/2})$ time. To the best of our knowledge, this is the first truly subquadratic time algorithm for building Single-Source DSOs on sparse graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Shortest paths; Theory of computation \rightarrow Data structures design and analysis; Theory of computation \rightarrow Cell probe models and lower bounds; Theory of computation \rightarrow Pseudorandomness and derandomization

Keywords and phrases derandomization, distance sensitivity oracle, single-source replacement paths, space lower bound

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.18

Related Version *Full Version*: <https://arxiv.org/abs/2106.15731>

Funding *Daide Bilò*: This work was partially supported by the Research Grant FBS2016_BILO, funded by “Fondazione di Sardegna” in 2016.



© Daide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 18; pp. 18:1–18:17

Leibniz International Proceedings in Informatics



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

1 Introduction

One of the basic problems in computer science is the computation of shortest paths and distances in graphs that are subject to a small number of transient failures. We study two central problems of this research area on undirected graphs G with n vertices and m edges, namely, the *Single-Source Replacement Paths* (SSRP) problem and *Single-Source Distance Sensitivity Oracles* (Single-Source DSOs).

The SSRP Problem. In the SSRP problem, we are given a graph G with a fixed source vertex s and are asked to compute, for every vertex t and edge e , the replacement distance $d(s, t, e)$, which is the length of the shortest s - t -path in the graph $G - e$, obtained by dropping the edge e . By first computing any shortest path tree for G rooted at s , one can see that there are only $O(n^2)$ relevant distances $d(s, t, e)$, namely, those for which e is in the tree.

Chechik and Cohen [10] presented an $\tilde{O}(m\sqrt{n} + n^2)$ time¹ combinatorial² SSRP algorithm for unweighted graphs. They also showed that the running time cannot be improved by polynomial factors, assuming that any combinatorial algorithm for Boolean Matrix Multiplication (BMM) on $n \times n$ matrices containing m 1's requires $mn^{1-o(1)}$ time. Gupta et al. [21] simplified the SSRP algorithm and generalized it to multiple sources. For a set of σ sources, they presented a combinatorial algorithm that takes $\tilde{O}(m\sqrt{n\sigma} + \sigma n^2)$ time. Grandoni and Vassilevska Williams [17, 18] gave an algorithm for both directed and undirected graphs with integer edge weights in the range $[1, M]$ that uses fast matrix multiplications and runs in $\tilde{O}(Mn^\omega)$ time, where $\omega < 2.37286$ is the matrix multiplication exponent [2, 25, 35]. We are only concerned with *positive* integer weights, but it is worth noting that SSRP with weights in $[-M, M]$ is strictly harder, modulo a breakthrough in Min-Plus Product computation, with a current best running time of $O(M^{0.8043} n^{2.4957})$ as shown by Gu et al. [20].

All the SSRP algorithms above are randomized, it is an interesting open problem whether they can be derandomized in the same asymptotic running time.

Single-Source DSOs. A Distance Sensitivity Oracle (DSO) is a data structure that answers queries (u, v, e) , for vertices u, v and edge e , by returning the replacement distance $d(u, v, e)$. A Single-Source DSO, with fixed source s , answers queries (t, e) with $d(s, t, e)$.

Of course, any SSRP algorithm gives a Single-Source DSO by just tabulating the whole output in $O(n^2)$ space, the replacement distances can then be queried in constant time. However, the space usage is far from optimal. Parter and Peleg [29] developed a deterministic algorithm that computes an $O(n^{3/2})$ size subgraph of G containing a breadth-first-search tree of $G - e$ for every failing edge e . The subgraph can also be thought of as a Single-Source DSO with $O(n^{3/2})$ space and query time. Bilò et al. [7] presented a Single-Source DSO of the same size with $\tilde{O}(\sqrt{n})$ query time and $\tilde{O}(mn)$ preprocessing time. Gupta and Singh [22] later designed a randomized Single-Source DSO of $\tilde{O}(n^{3/2})$ size, $\tilde{O}(mn)$ preprocessing time,³ but with a better $\tilde{O}(1)$ query time. The results in the latter two works generalize to the case of σ sources in such a way that the time and size scale by $o(\sigma)$ factors.

For the case of $\sigma = n$ sources, that is, general (all-pairs) DSOs, Bernstein and Karger [5, 6] designed an oracle taking $\tilde{O}(n^2)$ space with constant query time, even for directed graphs with real edge weights. The space was subsequently improved to $O(n^2)$ by Duan and Zhang [16],

¹ For a non-negative function $f = f(n)$, we use $\tilde{O}(f)$ to denote $O(f \cdot \text{polylog}(n))$.

² The term “combinatorial algorithm” is not well-defined, and is often interpreted as not using any matrix multiplication. Arguably, combinatorial algorithms can be considered efficient in practice as the constants hidden in the matrix multiplication bounds are rather high.

³ The authors of [22] do not report the preprocessing time, but it can be reconstructed as $\tilde{O}(mn)$.

which is optimal [34]. The combinatorial $\tilde{O}(mn)$ time preprocessing for building the DSOs is conditionally near-optimal as it matches the best known bound (up to polylogarithmic factors) for the simpler problem of finding the All-Pairs Shortest Paths (APSP). The conditional lower bound in [10], stating that there exists no combinatorial algorithm solving the undirected SSRP problem with real edge weights in $O(mn^{1-\varepsilon})$ time for any positive $\varepsilon > 0$, unless there is a combinatorial algorithm for the APSP problem in $O(mn^{1-\varepsilon})$ time, also implies that there exists no *Single-Source* DSO with $\tilde{O}(1)$ query time and $O(mn^{1-\varepsilon})$ preprocessing time for real edge weights. Therefore, the DSOs in [6, 16], are also conditionally near-optimal for the single source case with real edge weights.

Several algebraic all-pairs DSOs with subcubic preprocessing time have been developed in the last decade for graphs with integer edge weights in $[1, M]$ [9, 11, 17, 30, 37]. Very recently, Gu and Ren [19] presented a randomized DSO achieving a $O(Mn^{2.5794})$ preprocessing time with $O(1)$ query time, improving upon the one by Ren [30, 31] with an $\tilde{O}(Mn^{2.6865})$ preprocessing time. Those DSOs can also be used in the single-source case, but the requirement to store the information for all pairs forces them to take $\Omega(n^2)$ space [34]. The algebraic SSRP algorithm in [18], seen as a data structure, has a better preprocessing time than any known (general) DSO but also takes $O(n^2)$ space, which we have seen to be wasteful.

We are not aware of an algebraic Single-Source DSO that simultaneously achieves $o(n^2)$ space and has a better preprocessing time than their all-pairs counterparts. It is interesting whether we can construct space-efficient oracles faster when focusing on a single source.

Additional results on replacement paths and DSOs (for single or multiple failures and directed graphs) can be found in [3, 9, 12, 13, 14, 15, 18, 23, 24, 26, 27, 28, 32, 36]. The most efficient Single-Source DSOs in their respective settings are shown in Table 1 below.

1.1 Our Contribution

We research SSRP algorithms, Single-Source DSO data structures, and the connection between the two. Our first contribution is presented in Section 5. We derandomize the near-optimal combinatorial SSRP algorithm of Chechik and Cohen [10] for undirected, unweighted graphs and the algebraic algorithm of Grandoni and Vassilevska Williams [17, 18] for undirected graphs with integer weights in the range $[1, M]$. (The second algorithm can be found in the full version.) Both deterministic algorithms have the same asymptotic runtime as their randomized counterparts.

► **Theorem 1.** *There is a deterministic, combinatorial SSRP algorithm for undirected, unweighted graphs running in time $\tilde{O}(m\sqrt{n} + n^2)$ and a deterministic, algebraic SSRP algorithm for undirected graphs with integer weights in the range $[1, M]$ running in $\tilde{O}(Mn^\omega)$.*

We present in Section 3 a deterministic reduction from the problem of building a Single-Source DSO to SSRP on undirected graphs with small integer edge weights.

► **Theorem 2.** *Let G be an undirected graph with integer edge weights in the range $[1, M]$ and let s be the source vertex. Suppose we are given access to a shortest path tree T_s of G rooted in s and all values $d(s, t, e)$ for vertices t of G and edges e in T_s . There is a deterministic, combinatorial algorithm that in time $O(n^2)$ builds a Single-Source DSO of size $O(M^{1/2}n^{3/2})$ with $\tilde{O}(1)$ query time. The same statement holds for vertex failures if instead we are given access to the values $d(s, t, v)$ for all vertices t and v of G .*

The algorithm does not require access to the graph G itself. As there can be up to $O(n^2)$ relevant distances $d(s, t, e)$, the running time is linear in the input. If the algorithm additionally has access to G and is given $O(m\sqrt{Mn} + n^2)$ time, the Single-Source DSO also reports the replacement paths $P(s, t, e)$ in time $\tilde{O}(1)$ per edge. The query time of the oracle can be improved to $O(1)$ at the cost of increasing the size of the oracle to $O(M^{1/3}n^{5/3})$.

Plugging the deterministic SSRP algorithms of Theorem 1 into our reduction of Theorem 2, gives the following Single-Source DSOs as corollaries.

► **Theorem 3.** *There is a deterministic, combinatorial Single-Source DSO for undirected, unweighted graphs taking $O(n^{3/2})$ space, with $\tilde{O}(m\sqrt{n} + n^2)$ preprocessing time, and $\tilde{O}(1)$ query time. There is a deterministic, algebraic Single-Source DSO for undirected graphs with integer weights in the range $[1, M]$ taking $O(M^{1/2}n^{3/2})$ space, with $\tilde{O}(Mn^\omega)$ preprocessing time, and $\tilde{O}(1)$ query time.*

When comparing the results with other Single-Source DSO with $o(n^2)$ space, the preprocessing time of our combinatorial solution is better by a factor of \sqrt{n} compared to previous oracles [7, 22]. The preprocessing time of the algebraic part of Theorem 3 improves (ignoring polylogarithmic factors) by a factor of $n^{2.5794-\omega} > n^{0.2}$ over the current best algebraic (all-pairs) DSO [19]. See Table 1 for more details. In fact, we combine the efficient preprocessing of SSRP algorithms (seen as DSOs) with a compression scheme that achieves nearly-optimal space. To the best of our knowledge, Theorem 3 presents the first algebraic Single-Source DSO with $o(n^2)$ space that achieves a better performance than any all-pairs DSO. It is also the first space-efficient Single-Source DSO for graphs with small integer weights.

We further study lower bounds for Single-Source DSOs. Note that given an oracle whose preprocessing time is P and query time is Q , one can solve the SSRP problem in time $P + n^2 \cdot Q$ by building the DSO and running the queries (t, e) for every $t \in V, e \in E(T_s)$. Therefore, if $n^2 \cdot Q = O(P)$, the $mn^{1/2-o(1)} + \Omega(n^2)$ conditional⁴ time-lower bound for the SSRP problem [10], obtained by a reduction from BMM, implies the same lower bound for P . The preprocessing of our combinatorial oracle in Theorem 3 is thus nearly optimal. We further investigate how much a Single-Source DSO can be compressed. In contrast to [10], we obtain an *unconditional* space-lower bound using an argument from information theory.

► **Theorem 4.** *Any Single-Source DSO must take $\Omega(\min\{M^{1/2}n^{3/2}, n^2\})$ bits of space on at least one $O(n)$ -vertex graph with integer edge weights in the range $[1, M]$.*

A small gap remains between Theorems 2 and 4 as the space is bounded at $\Omega(M^{1/2}n^{3/2})$ bits, while the oracle takes this many machine words. Nevertheless, it shows that on dense graphs our Single-Source DSOs in Theorem 3 have near-optimal space.

The Single-Source DSOs presented above all have $\Omega(n^2)$ preprocessing time, which cannot be avoided for graphs with $m = \Omega(n^{3/2})$, assuming the BMM conjecture. SSRP algorithms require $\Omega(n^2)$ time simply to output the solution. It is not clear whether this lower bound also applies to Single-Source DSO on sparse graphs. We partially answer this question negatively by developing a truly subquadratic, randomized Single-Source DSO in Section 6. We use new algorithmic techniques and structural properties of independent interest.

► **Theorem 5.** *There is a randomized Single-Source DSO taking $O(M^{1/2}n^{3/2})$ space that has $\tilde{O}(1)$ query time w.h.p.⁵ The oracle also reports a replacement path in $\tilde{O}(1)$ time per edge w.h.p. On graphs with $m = O(M^{3/4}n^{7/4})$ edges, the preprocessing time is $\tilde{O}(M^{7/8}m^{1/2}n^{11/8})$. If the graph is sparse, meaning $m = O(n^{5/4-\varepsilon}/M^{7/4})$ for any $\varepsilon > 0$, this is $\tilde{O}(n^{2-\varepsilon/2})$.*

1.2 Comparison with Previous Work

Table 1 shows a comparison of the most efficient Distance Sensitivity Oracles in their respective setting, as well as the results presented in this work. We distinguish four dimensions of different problem types.

⁴ The $\Omega(n^2)$ term is unconditional and stems from the size of the output, see [10].

⁵ An event occurs *with high probability* (w.h.p.) if it has probability at least $1 - n^{-c}$ for some $c > 0$.

■ **Table 1** Comparison of results. [†]The preprocessing time is for graphs with $m = O(M^{3/4}n^{7/4})$.

Preprocessing time	Space	Query time	Setting	Reference
$\tilde{O}(mn)$	$O(n^2)$	$O(1)$	D C W Ap	[6, 16]
$\tilde{O}(Mn^{2.5794})$	$\tilde{O}(n^2)$	$O(1)$	R A I Ap	[19]
$\tilde{O}(mn^{1/2} + n^2)$	$O(n^2)$	$O(1)$	R C U Ss	[10]
$\tilde{O}(Mn^\omega)$	$O(n^2)$	$O(1)$	R A I Ss	[18]
$\tilde{O}(mn)$	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^{1/2})$	D C U Ss	[7]
$\tilde{O}(mn)$	$\tilde{O}(n^{3/2})$	$\tilde{O}(1)$	R C U Ss	[22]
$\tilde{O}(mn^{1/2} + n^2)$	$O(n^{5/3})$	$O(1)$	D C U Ss	Lemma 10
$\tilde{O}(mn^{1/2} + n^2)$	$O(n^{3/2})$	$\tilde{O}(1)$	D C U Ss	Theorem 3
$\tilde{O}(Mn^\omega)$	$O(M^{1/3}n^{5/3})$	$O(1)$	D A I Ss	Lemma 10
$\tilde{O}(Mn^\omega)$	$O(M^{1/2}n^{3/2})$	$\tilde{O}(1)$	D A I Ss	Theorem 3
$\tilde{O}(M^{7/8}m^{1/2}n^{11/8})^\dagger$	$O(M^{1/2}n^{3/2})$	$\tilde{O}(1)$	R C I Ss	Theorem 5

1. Randomized (R) vs. deterministic (D),
2. Combinatorial (C) vs. algebraic (A),
3. Unweighted (U) vs. real weights (W) vs. integer weights in $[1, M]$ (I),
4. All-Pairs (Ap) vs. single-source (Ss).

Our deterministic, combinatorial Single-Source DSO from Theorem 3 has near-optimal space, preprocessing and query time for dense graphs. It improves the preprocessing time of the randomized DSOs by Bernstein and Karger [6], Bilò et al. [7], and Gupta and Singh [22] by a factor of $O(\sqrt{n})$. When viewing the randomized SSRP algorithm of Chechik and Cohen as an oracle, our solution has the same preprocessing time but reduces the space requirement, by an near-optimal factor of $O(n^{1/3})$ while increasing the query time to only $\tilde{O}(1)$.

Our algebraic combinatorial Single-Source DSO from Theorem 3 has near-optimal space and query time for dense graphs, its preprocessing time improves over the randomized, algebraic DSOs of Chechik and Cohen [11], Ren [30, 31], as well as Gu and Ren [19] by a factor of $\tilde{O}(n^{2.5794-\omega})$. It has the same preprocessing time as the SSRP algorithm by Grandoni and Vassilevska Williams [18], but compresses the output to $O(M^{1/2}n^{3/2})$ space.

Our Single-Source DSO from Lemma 10 even achieves constant query time at the expense of larger $O(n^{5/3})$ (respectively, $O(M^{1/3}n^{5/3})$) space. All of our oracles can handle vertex failures and are path-reporting, the query time then corresponds to the time needed per edge of the replacement path. In Theorem 5, we also obtain Single-Source DSO with subquadratic preprocessing time for sparse graphs.

1.3 Techniques

Multi-stage derandomization. To derandomize the SSRP algorithms, we extend the techniques by Alon, Chechik, and Cohen [3] to identify a small set of critical paths we need to hit. In [3], a single set of paths was sufficient, we extend this to a hierarchical multi-stage framework. The set of paths in each stage depends on the hitting set found in the previous ones. For example, a replacement path from s to t avoiding the edge e decomposes into two shortest paths $P(s, q)$ and $P(q, t)$ in the original graph for some unknown vertex q , see [1]. It is straightforward to hit all of the components $P(s, q)$. We then use this hitting set in a more involved way to find sets of vertices that also intersect all of the subpaths $P(q, t)$.

Versatile compression. The key observation of our reduction to SSRP is that any shortest s - t -path can be partitioned into $O(\sqrt{Mn})$ segments such that all edges in a segment have the same replacement distance. Gupta and Singh [22] proved this for unweighted graphs. However, it is not obvious how to generalize their approach to the weighted case. We give a simpler proof in the presence of small integer weights, which immediately transfers also to vertex failures. We further show how to extend this to multiple targets and even reuse it to obtain the subquadratic algorithm on sparse graphs. In [22], a randomized oracle was presented that internally uses the rather complicated data structures of Demetrescu et al. [14]. We instead give a deterministic construction implementable with only a few arrays. Unfortunately, the compression scheme crucially depends on the graph being undirected.

Advanced search for replacement paths. The randomized algorithm building the DSO in subquadratic time for sparse graphs needs to find the $O(\sqrt{Mn})$ segments partitioning the s - t -path. Naively, this takes $O(n)$ time per target vertex t as we need to explore the whole path for potential segment endpoints and do not know the corresponding replacement paths in advance. We use standard random sampling to hit all such replacements paths with only a few vertices and exploit the path's monotonicity properties to develop more advanced search techniques. This reduces the time needed per target to $O(n^{1-\varepsilon})$, after some preprocessing. The analysis uses the fact that entire subpaths can be discarded without exploration.

Open problems. Our compression scheme and the randomized, subquadratic Single-Source DSO on sparse graphs can also handle vertex failures rather than only edge failures. It remains an open question whether one can obtain efficient deterministic SSRP algorithms in the vertex-failure scenario. If an analog of Theorem 1 held for vertex failures, then Theorem 2 would directly transfer the extension also to the DSOs of Theorem 3. Another interesting open question is whether there is a Single-Source DSO with deterministic, truly subquadratic time preprocessing on graphs with $m = O(n^{3/2-\varepsilon})$ edges. Can one obtain better Single-Source DSOs, and prove matching lower bounds, for sparse graphs?

2 Preliminaries

We let $G = (V, E, w)$ denote the undirected, edge-weighted base graph on n vertices and m edges, and tacitly assume $m \geq n$. The weights $w(e)$, $e \in E$, are integers in $[1, M]$ with $M = \text{poly}(n)$. For an undirected, weighted graph H , we denote by $V(H)$ the set of its vertices, and by $E(H)$ edge set of its edges. We write $e \in H$ for $e \in E(H)$ and $v \in H$ for $v \in V(H)$. Let P be a simple path in H . The *length* or *weight* $w(P)$ of P is $\sum_{e \in E(P)} w(e)$. For $u, v \in V(H)$, we denote by $P_H(u, v)$ a shortest path (one of minimum weight) from u to v . If a particular shortest path is intended, this will be made clear from the context. The *distance* of u and v is $d_H(u, v) = w(P_H(u, v))$. We drop the subscript when talking about the base graph G . The restriction on the maximum weight M allows us to store any graph distance in a single machine word on $O(\log n)$ bits. Unless explicitly stated otherwise, we measure space complexity in the number of words.

Let $x, y \in V(P)$ be two vertices on the simple path P . We denote by $P[x..y]$ the subpath of P from x to y . Let $P_1 = (u_1, \dots, u_i)$ and $P_2 = (v_1, \dots, v_j)$ be two paths in H . Their *concatenation* is $P_1 \circ P_2 = (u_1, \dots, u_i, v_1, \dots, v_j)$, provided that $u_i = v_1$ or $\{u_i, v_1\} \in E(H)$.

Fix some *source vertex* $s \in V$ in the base graph G . For any *target vertex* $t \in V$ and edge $e \in E$, we let $P(s, t, e)$ denote a *replacement path* for e , that is, a shortest path from s to t in G that does not use the edge e . Its weight $d(s, t, e) = w(P(s, t, e))$ is the *replacement*

distance. Given a specific shortest path $P(s, t)$ in G and a replacement path $P(s, t, e)$, we can assume w.l.o.g. that the latter is composed of the *common prefix* that it shares with $P(s, t)$, the *detour* part which is edge-disjoint from $P(s, t)$, and the *common suffix* after $P(s, t, e)$ remerges with $P(s, t)$. All statements apply to vertex failures as well.

3 Using SSRP to Build Single-Source DSOs

In this section, we prove Theorem 2. We describe how to deterministically reduce the task of building a Single-Source DSO to computing the replacement distances in the SSRP problem. Recall that we assume we are given a shortest path tree T_s of the base graph G rooted in the source s . This does not lose generality as we could as well compute it in time $O(m)$ via Thorup's algorithm [33]. However, the tree T_s focuses our attention to the $O(n^2)$ *relevant* replacement distances in G . The failure of an edge e can only increase the distance from s to some target t if e lies on the s - t -path $P(s, t)$ in T_s . Given a query (e, t) , we can thus check whether e is relevant for t in $O(1)$ time using a lowest common ancestor (LCA) data structure of size $O(n)$ [4]. If the maximum weight M is larger than n , we are done as we store the relevant replacement distances, original graph distances, and the LCA data structure.

However, for $M \leq n$, there are more space-efficient solutions. Using time $O(n^2)$, that is, linear in the number of relevant distances, we compress the space needed to store them down to $O(M^{1/2} n^{3/2})$ while increasing the query time only to $\tilde{O}(1)$. This scheme also allows several extensions, namely, handling vertex failures, reporting fault-tolerant shortest path trees, or retaining constant query time by using slightly more space. We first give an overview of the reduction. Suppose we have a set of *pivots* $D \subseteq V$ such that any s - t -path $P(s, t)$ in T_s has at least one element of D among its last \sqrt{n} vertices. For a target t , let x be the pivot on $P(s, t)$ that is closest to t . We distinguish three cases depending on the failing edge e .

- **Near case.** The edge e belongs to the near case if it is on the subpath $P(s, t)[x..t]$ from the last pivot to the target. We construct a data structure to quickly identify those edges. It is then enough to store the associated replacement distances explicitly.
- **Far case I.** The edge e belongs to the far case I if it is on the subpath $P(s, t)[s..x]$ and there is a replacement path for e that uses the vertex x . We handle this by storing a linear number of distances for every pivot in D .
- **Far case II.** We are left with edges e on $P(s, t)[s..x]$ for which no replacement path uses x . We show that there are only $O(M^{1/2} n^{3/2})$ many different replacement distances of this kind. We can find the correct distance in $\tilde{O}(1)$. This is the only case with a quadratic running time, space requirements depending on M , and a super-constant query time. We also show how to avoid the latter at the expense of a higher space complexity.

Near case. We first describe how to obtain the set D . We also take D to denote a representing data structure. That is, for all $t \in V$, $D[t]$ shall denote the last pivot on the path $P(s, t)$ in T_s . A deterministic greedy algorithm efficiently computes a small sets D .

► **Lemma 6.** *There exists a set $D \subseteq V$ with $|D| \leq \sqrt{n}$, computable in time $\tilde{O}(n)$, such that every s - t -path in T_s contains a pivot in D among its last \sqrt{n} vertices. In the same time, we can compute a data structure taking $O(n)$ space that returns $D[t]$ in constant time.*

Let $x = D[t]$ be the pivot assigned to t . An edge e belongs to the *near case* with respect to t if it lies on $P(x, t) = P(s, t)[x..t]$. Observe that $P(x, t)$ has less than \sqrt{n} edges. We store $d(s, t, e)$ for the near case in an array with (t, e) as key. With access to the distances, the array can be computed in $O(n^{3/2})$ total time and space. Consider a query (t, e) such that e has already been determined above to be on the path $P(s, t)$. The edge $e = \{u, v\}$ thus belongs to the near case iff $D[u] = D[v] = x$. If so, we look up $d(s, t, e)$ in the array.

Far case I. We say a query (t, e) belongs to the *far case* if e is on the subpath $P(s, x) = P(s, t)[s..x]$. These are the queries not yet handled by the process above. Note that $d(s, t, e) \leq d(s, x, e) + d(x, t)$ holds for all queries in the far case. If a replacement path $P(s, x, e)$ exists, $P(s, x, e) \circ P(s, t)[x..t]$ is some s - t -path that avoids e whose length is the right-hand side; otherwise, we have $d(s, x, e) = \infty$ and $d(s, t, e) \leq d(s, x, e) + d(x, t)$ holds vacuously. We split the far case depending on the existence of certain replacement paths. Recall that we can assume that any replacement path consists of a common pre- and suffix with the original path $P(s, t)$ and a detour that is edge-disjoint from $P(s, t)$. We let (t, e) belong to the *far case I* if e is on $P(s, x)$ and there is a replacement path $P(s, t, e)$ that uses the vertex x . It is readily checked that for a query in the far case this holds iff $d(s, t, e) = d(s, x, e) + d(x, t)$. Otherwise, that is, if no replacement path $P(s, t, e)$ uses x or, equivalently, $d(s, t, e) < d(s, x, e) + d(x, t)$, the query is said to be in the *far case II*.

It takes too much space to store the replacement distances for all edges in the far case, or memorize which edge falls in which subcase. Instead, we build two small data structures and, at query time, compute two (potentially different) distances. We show that always the smaller one is correct, which we return as the final answer. First, for every pivot $x \in D$ and edge $e \in P(s, x)$, we store the replacement distance $d(s, x, e)$. Since $|D| \leq \sqrt{n}$ and $|E(P(s, x))| \leq n$, we can do so in $O(n^{3/2})$ time and space. Given a query (t, e) in the far case, we access the storage corresponding to $D[t] = x$, retrieve $d(s, x, e)$, and add $d(x, t) = d(s, t) - d(s, x)$. This gives the first candidate distance. It may overestimate $d(s, t, e)$, namely, if e belongs to the far case II.

Far case II. This case is more involved than the previous. We make extensive use of what we call break points. Let e_1, \dots, e_k be the edges of $P(s, t)[s..x]$ in the far case II (w.r.t. t) in increasing distance from s . We then have $d(s, t, e_1) \geq \dots \geq d(s, t, e_k)$. This is due to the fact that any replacement path $P(s, t, e_i)$ avoids the whole subpath starting with e_i and ending in x . Its length is thus at least the replacement distance for any $e_j, j \geq i$. Let u_i be vertex of e_i that is closer to s . We say u_i is a *break point* if $d(s, t, e_i) > d(s, t, e_{i+1})$. A break point is the beginning of a segment in which the edges in the far case II have equal replacement distance. We show that there are only $O(\sqrt{Mn})$ break points/replacement distances.

For the analysis, we let the edges choose a *representative* replacement path. They do so one after another in the above order. Edge e_i first checks whether one of its replacement paths has previously been selected by an earlier edge $e_h, h < i$. If so, it takes the same one; otherwise, it chooses a possible replacement path arbitrarily. Let \mathcal{R} denote the set of representatives and let $R \in \mathcal{R}$. We define z_R to be the first vertex on the detour part of R . The vertices $z_R, R \in \mathcal{R}$, are also important for the subquadratic algorithm in Section 6.

► **Lemma 7.** *Edges e_i and e_j that belong to the far case II choose the same representative iff $d(s, t, e_i) = d(s, t, e_j)$. All representatives have different lengths and $|\mathcal{R}|$ equals the number of break points. Let $R, R' \in \mathcal{R}$ be such that R' is the next shorter representative after R . We have $d(s, z_R) < d(s, z_{R'})$ and all edges represented by R lie on the subpath $P(s, t)[z_R..z_{R'}]$. There is exactly one break point on $P(s, t)[z_R..z_{R'}]$, the one corresponding to length $w(R)$.*

Proof. Edges with different replacement distances have disjoint sets of replacement paths to choose from. Now suppose the replacement distances $d(s, t, e_i) = d(s, t, e_j)$ are equal. Without loosing generality, the edge $e_j, j \geq i$, is further away from s and selects its representative after e_i . The representative replacement path R for edge e_i also avoids e_j since it does not remerge with $P(s, t)$ prior to pivot x . As the distances $d(s, t, e_j) = d(s, t, e_i) = w(R)$ are the same, R is in fact a replacement path for e_j and is selected again as representative. The assertions of the different lengths and the total number of representatives easily follow.

Let $R \in \mathcal{R}$ be a representative replacement path. The first vertex z_R of its detour part must be closer to s than all edges it represents as R avoids them. Let e^* be the edge closest to s that belongs to the far case II and is represented by R . The break point u^* starting the segment with replacement distance $w(R)$ is thus the vertex of e^* that is closer to s .

Let now $R' \in \mathcal{R}$ be the next shorter representative after R . If we had $d(s, z_{R'}) \geq d(s, z_R)$, then R' would be a path that avoids e^* and has length $w(R') < w(R) = d(s, t, e^*)$ strictly smaller than the replacement distance, a contradiction. Reusing the same arguments as before, we also get that the break point corresponding to $w(R')$ lies after $z_{R'}$ and that the break point $u^* \in e^*$ cannot lie below $z_{R'}$ (on subpath $P(s, t)[z_{R'}..t]$). In summary, the subpath $P(s, t)[z_R..z_{R'}]$ contains exactly one break point, namely, u^* . ◀

It is left to prove that $|\mathcal{R}| = O(\sqrt{Mn})$. The following lemma is the heart of our compression scheme. It simplifies and thereby generalizes a result by Gupta and Singh [22] for unweighted undirected graphs. The argument we use is versatile enough to not only cover integer-weighted graphs, it extends to vertex failures as well (Lemma 9). A similar idea also allows us to design an oracle with constant query time (Lemma 10) and the subquadratic preprocessing algorithm on sparse graphs (Theorem 5). Unfortunately, the argument crucially depends on the graph being undirected. New techniques are needed to compress the fault-tolerant distance information in directed graphs.

▶ **Lemma 8.** *The number of representatives for edges on $P(s, t)$ is $|\mathcal{R}| \leq 3\sqrt{Mn}$.*

Proof. All representatives are of different length by Lemma 7. Also, they have length at least $d(s, t)$, the weight of the original s - t -path P . Hence, there are only $2\sqrt{Mn}$ many of length at most $d(s, t) + 2\sqrt{Mn}$. We now bound the number of *long* representatives, which are strictly longer than that. Let R be a long representative. Its detour part is longer than $2\sqrt{Mn}$, whence it must span at least $2\sqrt{n/M}$ vertices. Consider the path on the first $\sqrt{n/M}$ vertices of the detour starting in z_R , we call it Stub_R . If Stub_R does not intersect with $\text{Stub}_{R'}$ for any other long $R' \in \mathcal{R}$, $R' \neq R$, there can only be $n/(\sqrt{n/M}) = \sqrt{Mn}$ stubs in total and thus as many long representatives.

To reach a contradiction, assume the stubs of R and R' intersect. Let e be an edge represented by R and $y \in V(\text{Stub}_R) \cap V(\text{Stub}_{R'})$ a vertex on both stubs. W.l.o.g. R' is strictly shorter than R and thus $z_{R'}$ comes behind z_R on the path P and e is on $P[z_R..z_{R'}]$ (Lemma 7). Note that $w(\text{Stub}_R), w(\text{Stub}_{R'}) \leq \sqrt{Mn}$. Therefore, the path $P^* = P[s..z_R] \circ R[z_R..y] \circ R'[y..z_{R'}] \circ P[z_{R'}..t]$ avoids e and has length $w(P^*) \leq d(s, t) + w(R[z_R..y]) + w(R'[y..z_{R'}]) \leq d(s, t) + 2\sqrt{Mn} < w(R)$. This is a contradiction to R being the representative of e . ◀

Observe how the argument in the proof above depends on the fact that we can traverse the segment $R'[z_{R'}..y] \subseteq \text{Stub}_{R'}$ in both directions. When following R' from s to t , we visit $z_{R'}$ prior to y , while for P^* it is the other way around. This is not necessarily true in a directed graph. Indeed, one can construct examples that have a directed path on $\Omega(n)$ edges in which each of them has its own replacement distance.

With access to the replacement distances, all break points can be revealed by a linear scan of the path $P(s, t)$ in time $O(n)$. Let $z_{i_1}, \dots, z_{i_{|\mathcal{R}|}}$ be the break points ordered by increasing distance to the source s and $e_{i_1}, \dots, e_{i_{|\mathcal{R}|}}$ the corresponding edges. For the data structure, we compute an ordered array of the original distances $d(s, z_{i_1}) < \dots < d(s, z_{i_{|\mathcal{R}|}})$ associated with the replacement distances $d(s, t, e_{i_j})$, taking $O(\sqrt{Mn})$ space. Let (e, t) be a query with $e = \{u, v\}$. We compute the index $j = \arg \max_{1 \leq k \leq |\mathcal{R}|} \{d(s, z_{i_k}) \leq d(s, u)\}$, with a binary search on the array in $O(\log n)$ time and retrieve $d(s, t, e_{i_j})$ as the second candidate distance.

The edge e lies on the subpath $P(s, t)[z_{i_j}..z_{i_{j+1}}]$ (respectively, on $P(s, t)[z_{i_{|\mathcal{R}|}}..x]$ if $j = |\mathcal{R}|$). It thus has replacement distance *at most* $d(s, t, e_{i_j})$. If e belongs to the far case II, the second candidate distance is exact and (strictly) smaller than the first one $d(s, x, e) + d(x, t)$; otherwise, the first candidate is smaller (or equal) and correct.

Scaling this solution to all targets $t \in V$ gives a total space requirement of $O(M^{1/2}n^{3/2})$. However, the preprocessing time is $O(n^2)$, dominated by the linear scans for each target.

3.1 Extensions

There are several possible extensions for our Single-Source DSO. While the transfer to vertex failures comes for free, reducing the query time to a constant, making the oracle path-reporting, or returning the whole fault-tolerant shortest path tree incurs additional costs of a higher space requirement or preprocessing time, respectively. We still assume the setting of Theorem 2, i.e., oracle access to the replacement distances for failing edges/vertices.

Vertex failures. The solutions for the near case, and far case I hold verbatim also for vertex failures. A vertex on the path $P(s, t)[s..x]$, except s itself, belongs to the *far case II* iff it satisfies $d(s, t, v) < d(s, t, x) + d(x, t)$. Let \mathcal{R}_V be the sets of representatives, now chosen by the *vertices*. The advantage of the proof of Lemma 8 is that it easily transfers to vertex failures. While the stubs of the detours may no longer be unique, they now intersect at most one other stub and identify *pairs* of representatives.

► **Lemma 9.** *The number of representatives for vertices on $P(s, t)$ is $|\mathcal{R}_V| \leq 5\sqrt{Mn}$.*

Constant query time. If we could query the break point of an edge in the far case II in $O(1)$ time, our Single-Source DSO had a constant overall query time. However, since the break points also depend on the target t , hard-coding them would yield a $O(n^2)$ space solution, which is wasteful for $M = o(n)$. Instead, we improve the analysis in Lemma 8. It hardly made any use of the fact that the pivot x is among the last \sqrt{n} vertices on the s - t -path in T_s and considered only a single target. We now strike a balance between selecting more pivots and grouping targets with the same assigned pivot together.

► **Lemma 10.** *There is an algorithm that, when given oracle access to the replacement distances for failing edges (vertices), preprocesses in $O(n^2)$ time a Single-Source DSO for edge (vertex) failures taking $O(\min\{M^{1/3}n^{5/3}, n^2\})$ space and having constant query time.*

Path-reporting oracles. We can adapt our Single-Source DSOs to also report the replacement paths using the same space. However, to do so it is not enough to have access to the replacement distances as the paths depend on the structure of G . Also, making the oracle path-reporting increases in preprocessing time, which now also depends on m .

► **Lemma 11.** *With access to G , there is a path-reporting Single-Source DSO for edge (vertex) failures with $O(\min\{m\sqrt{Mn}, mn\} + n^2)$ preprocessing time and either $O(\min\{M^{1/2}n^{3/2}, n^2\})$ space and $\tilde{O}(1)$ query time per edge, or $O(\min\{M^{1/3}n^{5/3}, n^2\})$ space and $O(1)$ query time.*

Fault-tolerant shortest path tree oracles. We are going one step further in the direction of fault-tolerant subgraphs, see for example [8, 29]. We enable our oracle to report, for any failing edge or vertex, the whole fault-tolerant single-source shortest path tree. Compared to the path-reporting version, we make sure to return every tree edge only once.

► **Lemma 12.** *With access to G , there is a data structure with $O(\min\{m\sqrt{Mn}, mn\} + n^2)$ preprocessing time, taking $O(\min\{M^{1/2}n^{3/2}, n^2\})$ space that, upon query $e \in E$ (respectively, $v \in V$), returns a shortest path tree for $G - e$ (respectively, $G - v$) rooted in s in time $O(n)$.*

4 Space Lower Bound

We now present an information-theoretic lower bound showing that the space of the Single-Source DSO resulting from our reduction is optimal up to the word size.

► **Theorem 4.** *Any Single-Source DSO must take $\Omega(\min\{M^{1/2}n^{3/2}, n^2\})$ bits of space on at least one $O(n)$ -vertex graph with integer edge weights in the range $[1, M]$.*

Proof. Let $M' = \min\{M, n\}$. We give an incompressibility argument in that we show that one can store any binary $n \times n$ matrix X across $\sqrt{n/M'}$ Single-Source DSOs. Not all of them can use only $o(\sqrt{M'}n^{3/2})$ bits of space as otherwise this would compress X to $o(n^2)$ bits. We create graphs $G_1, G_2, \dots, G_{\sqrt{n/M'}}$. Each of them has $O(n)$ vertices and maximum edge weight M . The graph G_k will be used to store the $\sqrt{M'n}$ rows of X with indices from $(k-1)\sqrt{M'n} + 1$ to $k\sqrt{M'n}$.

We first describe the parts that are common to all of the G_k . Let $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ be two sets of n vertices each, we connect a_i and b_j by an edge of weight 1 iff $X[i, j] = 1$. There are no other edges between A and B . We also add a path $P = (v_1, \dots, v_{\sqrt{M'n}})$ all of whose edges have weight 1. The vertex $s = v_{\sqrt{M'n}}$ is the source in each graph. Also, let $\{v_0, v_1\}$ be an edge of weight M , it serves to raise the maximum edge weight to M , if needed. Specifically in G_k and for each $1 \leq i \leq \sqrt{M'n}$, we connect the vertex v_i with $a_{(k-1)\sqrt{M'n}+i}$ by a path $P_{k,i}$ of total weight $2i - 1$. Due to the edge weights, we can make the path $P_{k,i}$ so that it uses at most $2i/M'$ edges and thus so many new vertices. In total, G_k has at most $2n + (\sqrt{M'n} + 1) + \sum_{i=1}^{\sqrt{M'n}} \frac{2i}{M'} = O(n)$ vertices due to $M' \leq n$.

Let e_i denote the edge $\{v_{i-1}, v_i\}$ on P . We claim that $X[(k-1)\sqrt{M'n} + i, j] = 1$ if and only if the replacement distance in G_k is $d_{G_k}(v_{\sqrt{M'n}}, b_j, e_i) = \sqrt{M'n} + i$. We assume $k = 1$, larger k follow in the same fashion. Observe that one has to go through a vertex in $A' = \{a_i, a_{i+1}, \dots, a_{\sqrt{M'n}}\}$ to reach b_j from the source $s = v_{\sqrt{M'n}}$. Conversely, A' is the only part of A that is reachable from s in $G_1 - e_i$ without using any vertex of B .

If there is no replacement path from s to b_j avoiding e_i , we have $d_{G_1}(s, b_j, e_i) = \infty$ and $X[i', j] = 0$ for all $i \leq i' \leq \sqrt{M'n}$, as desired. Let thus $P(s, b_j, e_i)$ be a replacement path and further a_{i^*} its first vertex that is in A (the one closest to the source s). Therefore, $i^* \geq i$ and $P(s, b_j, e_i)$ has the form $(v_{\sqrt{M'n}}, \dots, v_{i^*}) \circ P_{1,i^*} \circ P'$ for some a_{i^*} - b_j -path P' . It holds that $d_{G_1}(v_{\sqrt{M'n}}, b_j, e_i) = (\sqrt{M'n} - i^*) + (2i^* - 1) + w(P') = \sqrt{M'n} + i^* - 1 + w(P') \geq \sqrt{M'n} + i$. Equality holds only if $i^* = i$ and $w(P) = 1$, thus a_i must be a neighbor of b_j and $X[i, j] = 1$ follows; otherwise, the replacement distance is strictly larger. ◀

5 Derandomizing Single-Source Replacement Paths Algorithms

In this section, we derandomize the combinatorial $\tilde{O}(m\sqrt{n} + n^2)$ time algorithm for SSRP of Chechik and Cohen [10] obtaining the same asymptotic running time. In the full version, we also derandomize the algebraic SSRP algorithm of Grandoni and Vassilevska Williams. When combined with the reduction of Section 3, they give deterministic Single-Source DSOs.

Suppose the base graph $G = (V, E)$ is unweighted. It follows from a result by Afek et al. [1, Theorem 1] that for every target $t \in V$, edge $e \in E$, and replacement path $P(s, t, e)$ in $G - e$, there exists a vertex q on $P(s, t, e)$ such that both subpaths $P(s, t, e)[s..q]$ and $P(s, t, e)[q..t]$

are shortest paths in the original graph G . Computing the vertex q directly for each pair (t, e) is too expensive. Instead, the algorithm in [10] employs a random hitting set for the subpaths. The only randomization used in [10] is to sample every vertex independently with probability $O((\log n)/\sqrt{n})$ to create a set $B \subseteq V$ of so-called *pivots*. The set B contains $\tilde{O}(\sqrt{n})$ such pivots w.h.p. The correctness of the algorithm relies on the following important property. With high probability, there exists a vertex $x \in B \cup \{s\}$ before q on $P(s, t, e)$ and a vertex $y \in B \cup \{t\}$ after q such that the subpath of $P(s, t, e)[x..y]$ has length only $\tilde{O}(\sqrt{n})$. Here, we describe how to compute the set B deterministically with the same properties. We defer the proof of correctness of the algorithm to the full version.

We derandomize the vertex selection using an approach similar to the one of Alon, Chechik, and Cohen [3]. Given paths D_1, \dots, D_k , where each contains at least L vertices, the folklore greedy algorithm constructs a hitting set of size $\tilde{O}(n/L)$, by iteratively covering the maximum number of unhit paths, in $\tilde{O}(kL)$ time. The challenge is to quickly compute a suitable set of paths. We construct three systems of path \mathcal{L}_1 , \mathcal{L}_2 , and \mathcal{L}_3 to obtain B .

We prepare some notation. For a rooted tree T , a vertex $v \in V(T)$, and an integer parameter $L \geq 0$, let $\text{Last}_{T,L}(v)$ be the subpath containing the last L edges of the path in the tree T from the root to v , or the whole path if it has length less than L . Let $|\text{Last}_{T,L}(v)|$ denote the number of edges on the path.

- **Paths \mathcal{L}_1 and hitting set B_1 .** Set \mathcal{L}_1 contains the last $\sqrt{n}/2$ edges of every path in T_s , $\mathcal{L}_1 = \{\text{Last}_{T_s, \sqrt{n}/2}(v) \mid v \in V, |\text{Last}_{T_s, \sqrt{n}/2}(v)| = \sqrt{n}/2\}$. As an alternative, we can also use Lemma 6 to compute in $\tilde{O}(n)$ time a deterministic hitting set B_1 for \mathcal{L}_1 of size $2\sqrt{n}$.
- **Paths \mathcal{L}_2 and hitting set B_2 .** We run a breadth-first search from every vertex $x \in B_1$ to compute the shortest paths trees T_x rooted in x , and define the second set to be $\mathcal{L}_2 = \{\text{Last}_{T_x, \sqrt{n}/2}(y) \mid x \in B_1, y \in V, |\text{Last}_{T_x, \sqrt{n}/2}(y)| = \sqrt{n}/2\}$. Greedy selection computes a hitting set B_2 for \mathcal{L}_2 of size $\tilde{O}(\sqrt{n})$ in total time $\tilde{O}(n^2)$.

Before we can define \mathcal{L}_3 , we need additional notation. Let $e = \{u, v\}$ be an edge in T_s such that u is closer to s than v and let $T_{s,v}$ be the subtree of T_s rooted in v . Let further $G_e = (V_e, E_e, w_e)$ be a weighted graph such that V_e contains s and the vertices $x \in V(T_{s,v})$ with $d(s, x) \leq d(s, v) + 4\sqrt{n}$. The edges of G_e that are inside of $T_{s,v}$ are the same as in G , and additionally every shortest path $P(s, x)$ from s to every vertex $x \in V_e$ such that $P(s, x)$ passes only through vertices outside of V_e (except for its first vertex s and its last vertex $x \in V_e$) is replaced with a shortcut edge (s, x) whose weight is equal to the length $d(s, x)$ of the corresponding shortest path $P(s, x)$, preserving the original paths distances (using weights). The SSRP algorithm in [10] computes Dijkstra's algorithm from s in each G_e . We let T_{G_e} denote the resulting shortest path tree.

- **Paths \mathcal{L}_3 and hitting set B_3 .** The third set $\mathcal{L}_3 := \{\text{Last}_{T_{G_e}, \sqrt{n}/2}(x) \mid x \in V, e \in E(T_s), |\text{Last}_{T_{G_e}, \sqrt{n}/2}(x)| = \sqrt{n}/2\}$ contains $O(n^{3/2})$ paths as every vertex $x \in V$ belongs to at most $4\sqrt{n}$ graphs G_e . We thus get a hitting set B_3 of size $\tilde{O}(\sqrt{n})$ in time $\tilde{O}(n^2)$. The deterministic set $B = B_1 \cup B_2 \cup B_3$ can then be used as pivots in the SSRP algorithm.

6 Subquadratic Preprocessing on Sparse Graphs

Finally, we show how to obtain a Single-Source DSO with subquadratic preprocessing at least on sparse graphs. In order to prove Theorem 5, we present an algorithm running in time $\tilde{O}(M^{7/8} m^{1/2} n^{11/8} + \frac{M^{1/8} m^{3/2}}{n^{3/8}})$. If $m = O(M^{3/4} n^{7/4})$, then the dominating term is $\tilde{O}(M^{7/8} m^{1/2} n^{11/8})$. If the graph even satisfies $m = O(n^{5/4-\varepsilon}/M^{7/4})$ for any $\varepsilon > 0$, then

the preprocessing time is $\tilde{O}(n^{2-\varepsilon/2})$. We explain the main part of the randomized algorithm that allows us to design the Single-Source DSO. The algorithm is easily adaptable to deal with vertex failures as well. The proofs and some of the technical details are deferred to the full version due to the lack of space. In the following, we assume that the graph is indeed sparse, that is, $m = O(n^{5/4-\varepsilon}/M^{7/4})$. The next sampling lemma is folklore, see e.g. [18, 32].

► **Lemma 13.** *Let H be a graph with n vertices, $c > 0$ a positive constant, and L such that $L \geq c \ln n$. Define a random set $R \subseteq V$ by sampling each vertex to be in R independently with probability $(c \ln n)/L$. Then, with probability at least $1 - \frac{1}{n^c}$, the size of R is $\tilde{O}(n/L)$. Let further \mathcal{P} be a set of ℓ simple paths in H , each of which spans at least L vertices. Then, with probability at least $1 - \frac{\ell}{n^c}$, we have $V(P) \cap R \neq \emptyset$ for every $P \in \mathcal{P}$.*

We employ random sampling to hit one shortest path on at least $L = \frac{n^{11/8}}{M^{1/8} m^{1/2}}$ edges for every pair of vertices. Any vertex is included in the set R of *random pivots* independently with a probability of $(3 \ln n)/L$. We also include the source s in R to hit all short s - t -paths. By Lemma 13, we have $|R| = \tilde{O}(n/L) = \tilde{O}(\frac{M^{1/8} m^{1/2}}{n^{3/8}})$ w.h.p. Randomization is used here since it takes too long to handle the $O(n^2)$ paths explicitly.

We additionally construct a set D of (possibly different, regular) *pivots* that are used to classify replacement paths into near case, far case I, and far case II similar to Section 3. The set D is computed deterministically using Lemma 6, where we select a pivot every \sqrt{n} levels. For a target vertex $t \neq s$, the *proper pivot* of t shall be that pivot $x \in D$ on the path $P(s, t)$ in T_s that is closest to t but satisfies $d(x, t) \geq 4ML$, or $x = s$ if there is no such pivot. We let $D_1[t]$ denote the proper pivot of t and $D_2[t] = D_1[D_1[t]]$, provided that $D_1[t] \neq s$.

For every random pivot $\chi \in R$ and every edge e on the path $P(s, \chi)$, we compute $d(s, \chi, e)$ in $\tilde{O}(m)$ time per pivot using the algorithm of Malik, Mittal, and Gupta [26]. In the same time bound, we also get the vertex of $P(s, \chi)$ at which $P(s, \chi, e)$ diverges and we assume that $P(s, \chi, e)$ represents the path that diverges from $P(s, \chi)$ at a vertex that is as close as possible to s .⁶ For each pivot $x \in D$ and every e on $P(s, x)$, we also compute $d(s, x, e)$. This takes total time $\tilde{O}(m(|D| + |R|)) = \tilde{O}(mn^{1/2} + \frac{M^{1/8} m^{3/2}}{n^{3/8}}) = \tilde{O}(\frac{m^{1/2} n^{9/8-\varepsilon/2}}{M^{7/8}} + \frac{M^{1/8} m^{3/2}}{n^{3/8}})$ and allows us to answer replacement distance queries in $O(1)$ time if the target is in $D \cup R$.

We are left to handle non-pivot targets. Fix a $t \in V \setminus (D \cup R)$ and let $x_1 = D_1[t]$, and $x_2 = D_2[t]$. We use similar cases as before.

- **Near case.** The edge e is on $P(s, t)[x_2..t] = P(x_2, t)$.
- **Far case I.** The edge e is on $P(s, t)[s..x_2] = P(s, x_2)$ and there is a replacement path $P(s, t, e)$ that passes through x_2 .
- **Far case II.** The edge e is on $P(s, x_2)$ and there is no replacement path $P(s, t, e)$ that passes through x_2 .

In the remainder, we show how to efficiently compute the replacement distances in the far case II as previously this was the only case with quadratic run time. The technical details of the near case are reported in full version. A shortest path tree of G and the replacement distances to targets in D are enough to handle the far case I, see Section 3.

Since in the far case II the pivot x_2 lies on $P(s, t)$, we can assume $P(s, t)$ to have length $d(s, t) \geq d(x_2, t) \geq 4ML$ and at least $4L$ edges. In the following, we use different indexing schemes pointing to objects and distances related to $P(s, t)$, all of them are ordered from the source s to pivot x_2 . First, we denote by R_1, \dots, R_k the k representative replacement paths

⁶ The replacement path $P(s, \chi, e)$ computed in [26] is obtained as the concatenation of a subpath $P(s, u)$ of T_s , an edge $\{u, v\}$ of $G - e$, and a subpath $P(v, \chi)$ in T_χ (the shortest paths tree of G rooted at χ).

for edges in the far case II. We have $k \leq 3\sqrt{Mn}$ by Lemma 8. Let the *distinguished* edge $e_\ell^* \in P(s, t)$ be the one that is closest to s such that R_ℓ represents e_ℓ^* , i.e., R_ℓ is a replacement path in $G - e_\ell^*$ and we fall in far case II. Set $d_\ell = w(R_\ell)$. As no replacement path from s to t for edge e_ℓ^* uses vertex x_2 , we have $d_\ell < d(s, x_2, e_\ell^*) + d(x_2, t)$. The distinguished edges e_1^*, \dots, e_k^* are ordered by increasing distance from s , this implies $d_1 > \dots > d_k$ for their replacement distances, see Section 3. Furthermore, let N be the number of *all* edges (of the far cases I and II) on the path $P(s, x_2) = (e_1, e_2, \dots, e_N)$, seen in order from s to x_2 . This way, we identify $P(s, x_2)$ with the interval $[1, N]$, an index $j \in [1, N]$ stands for the j -th edge e_j on $P(s, x_2)$. With a slight abuse of notation, we also say that $e_j \in [a, b]$ in case $j \in [a, b]$.

We employ the random pivots to efficiently compute all the k pairs (d_ℓ, e_ℓ^*) w.h.p. The key idea is that, for each failing edge e on $P(s, x_2)$, there exists w.h.p. a random pivot $\chi \in R$ such that $d(\chi, t) \leq ML$ and $d(s, t, e) = d(s, \chi, e) + d(\chi, t)$ simultaneously hold. To see this, recall that any replacement path $P(s, t, e)$ has at least $4L$ edges and let y be the vertex such that $P(s, t, e)[y..t]$ consists of the last L of them. We claim that $P(s, t, e)[y..t]$ is in fact a shortest path in G . Assume there were a shorter y - t -path, then it must contain e and have length at least $d(x_2, t) \geq 4ML$, a contradiction. Therefore, *some* shortest y - t -path in G has at least L edges and is thus hit by a random pivot χ w.h.p., which gives the equality. Any reference to high probability refers to this fact. We use it to design a recursive algorithm that finds the pairs (d_ℓ, e_ℓ^*) w.h.p. in time $O(|R|M^{3/4}n^{3/4}) = \tilde{O}(M^{7/8}m^{1/2}n^{3/8})$ per target.

Recall that we view $P(s, x_2)$ as $[1, N]$. When exploring a subinterval $[a, b]$, the algorithm searches for a pair (d_ℓ, e_ℓ^*) with a distinguished edge $e_\ell^* \in [a, b]$. The algorithm knows both an upper bound $\Delta_{[a, b]}$ and a lower bound $\delta_{[a, b]}$ on the admissible values for d_ℓ . More precisely, $\Delta_{[a, b]} + 1$ corresponds w.h.p. to the smallest possible value $d_{\ell'}$ such that $e_{\ell'}^* \in [1, a-1]$ (the lower the index, the higher is $d_{\ell'}$); similarly, $\delta_{[a, b]} - 1$ is the the largest possible value $d_{\ell'}$ for $e_{\ell'}^* \in [b+1, N]$. In the beginning, we set $\Delta_{[1, N]} = \infty$, $\delta_{[1, N]} = 0$ and the algorithm explores the entire interval $[1, N]$. It terminates when there are no more unexplored subintervals.

We now describe the search for d_ℓ with $e_\ell^* \in [a, b]$. We assume $a \leq b$ and $\delta_{[a, b]} \leq \Delta_{[a, b]}$ as otherwise no such pair exists. Set $\mu = \max_{j \in [a, b]} \{d(s, x_2, e_j) + d(x_2, t)\}$. The algorithm keeps searching in the interval only if $\mu > \delta_{[a, b]}$. Indeed, if $\mu \leq \delta_{[a, b]}$, we know for sure that such a pair does not exist as there must be a replacement path (of type far case I) that passes through vertex x_2 . We first compute the largest index $j \in [a, b]$ for which $\mu = d(s, x_2, e_j) + d(x_2, t)$. We do so by employing a range minimum query (RMQ) data structure to support such queries in constant time after an $O(N) = O(n)$ time preprocessing [4]. Observe that the same data structure can be reused for all the target vertices t' for which $D_2[t'] = x_2$. It is enough that it stores the values $d(s, x_2, e)$, instead of $d(s, x_2, e) + d(x_2, t)$. The former distances are independent of the considered target and we already computed them above. We use only $O(|D|)$ RMQ data structures, which we prepare in $O(n|D|) = O(n^{3/2})$ time.

In the following, we assume $\mu > \delta_{[a, b]}$. We select a candidate replacement path for e_j by choosing the shortest one that runs through a random pivot in $O(|R|)$ time via brute-force search in the data we computed above for the targets in R . Ties are broken in favor of the replacement path $P(s, \chi, e_j)$ that diverges from $P(s, x_2)$ at the vertex that is closest to s . Let $\delta = \min_{\chi \in R} \{d(s, \chi, e_j) + d(\chi, t)\}$ be the length of such a replacement path, w.h.p. it is the actual replacement distance $P(s, t, e_j)$. Let further χ_j be the minimizing random pivot, and z_j the vertex of $P(s, x_2)$ at which $P(s, \chi_j, e_j)$ diverges. We check whether $\delta < \mu$ and $\delta \leq \Delta_{[a, b]}$ holds. If either of the two conditions is violated, then there is no need to keep searching in the interval $[a, j]$, as shown in the next lemma. In this case, the algorithm makes a recursive call on the *lower* interval $[j+1, b]$ (the one with smaller replacement distances) by setting $\Delta_{[j+1, b]} = \Delta_{[a, b]}$ and $\delta_{[j+1, b]} = \delta_{[a, b]}$. We say that the search was *unsuccessful*.

► **Lemma 14.** *If $\delta \geq \mu$ or $\delta > \Delta_{[a,b]}$, then, w.h.p. we have $e_\ell^* \notin [a, j]$ for all $\ell \in [k]$.*

Suppose the search is *successful*, that is, $\delta < \mu$ and $\delta \leq \Delta_{[a,b]}$. We then use binary search techniques⁷ to compute in $O(\log n)$ time the smallest index $i \in [a, j]$ for which the edge e_i lies on the subpath $P(s, x_2)[z_j..x_2]$ and $\delta < d(s, x_2, e_i) + d(x_2, t)$ holds. The case $i = j$ is possible. The condition on e_i is such that $P(s, x_j, e_j)$ also avoids e_i , which implies $d(s, t, e_i) \leq \delta < d(s, x_2, e_i) + d(x_2, t)$. The edge e_i must belong to the far case II w.r.t. target t . We show that in fact (δ, e_i) is w.h.p. the sought pair with $e_\ell^* \in [a, j]$ and minimum d_ℓ .

► **Lemma 15.** *Let $\ell \in [k]$ be maximal such that $e_\ell^* \in [a, j]$. Then, w.h.p. $\delta = d_\ell$ and $e_i = e_\ell^*$.*

The algorithm outputs (δ, e_i) and recurses on the lower interval $[j+1, b]$ with new bounds $\Delta_{[j+1,b]} = \delta - 1$ and $\delta_{[j+1,b]} = \delta_{[a,b]}$, as well as on the *upper* interval $[a, i-1]$, with $\Delta_{[a,i-1]} = \Delta_{[a,b]}$ and $\delta_{[a,i-1]} = \delta + 1$. This is justified since the edges in $[i, j]$ that belong to the far case II are w.h.p. precisely the ones represented by the path R_ℓ of length $d_\ell = \delta$.

The time needed for one target t is proportional (up to a log-factor) to the number of random pivots and the overall number of searches. There are $k = O(\sqrt{Mn})$ successful searches by Lemma 8. The following lemma bounds the number of unsuccessful searches.

► **Lemma 16.** *The number of unsuccessful searches for a single target vertex is $O(M^{3/4}n^{3/4})$.*

The algorithm computes w.h.p. all pairs for one target vertex in time $\tilde{O}(|R|M^{3/4}n^{3/4}) = \tilde{O}(M^{7/8}m^{1/2}n^{3/8})$, scaling this to all targets gives $\tilde{O}(M^{7/8}m^{1/2}n^{11/8})$.

References

- 1 Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by Path Concatenation: Fast Recovery of MPLS Paths. *Distributed Computing*, 15:273–283, 2002. doi:10.1007/s00446-002-0080-6.
- 2 Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.
- 3 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic Combinatorial Replacement Paths and Distance Sensitivity Oracles. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.
- 4 Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium Theoretical Informatics (LATIN)*, pages 88–94, 2000. doi:10.1007/10719839_9.
- 5 Aaron Bernstein and David R. Karger. Improved Distance Sensitivity Oracles via Random Sampling. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, pages 34–43, 2008. URL: <https://dl.acm.org/citation.cfm?id=1347082.1347087>.
- 6 Aaron Bernstein and David R. Karger. A Nearly Optimal Oracle for Avoiding Failed Vertices and Edges. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.
- 7 Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient Oracles and Routing Schemes for Replacement Paths. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 13:1–13:15, 2018. doi:10.4230/LIPIcs.STACS.2018.13.

⁷ Let interval $[a', b'] \subseteq [a, j]$ lie entirely below z_j . We divide it into subintervals $[a', j']$ and $[j'+1, b']$ of roughly equal sizes and check whether the maximum value returned by the RMQ data structure on query $[a', j']$ is still larger than δ . If so, we recurse on the interval $[a', j']$; otherwise, on $[j'+1, b']$.



- 8 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-Tolerant Approximate Shortest-Path Trees. *Algorithmica*, 80:3437–3460, 2018. doi:10.1007/s00453-017-0396-z.
- 9 Jan van den Brand and Thatchaphol Saranurak. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS, 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.
- 10 Shiri Chechik and Sarel Cohen. Near Optimal Algorithms for the Single Source Replacement Paths Problem. In *Proceedings of the 30th Annual Symposium on Discrete Algorithms (SODA)*, pages 2090–2109, 2019. doi:10.1137/1.9781611975482.126.
- 11 Shiri Chechik and Sarel Cohen. Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time. In *Proceedings of the 52nd Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020. doi:10.1145/3357713.3384253.
- 12 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f -Sensitivity Distance Oracles and Routing Schemes. *Algorithmica*, 63:861–882, 2012. doi:10.1007/s00453-011-9543-0.
- 13 Shiri Chechik and Ofer Magen. Near Optimal Algorithm for the Directed Single Source Replacement Paths Problem. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 81:1–81:17, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.
- 14 Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. Oracles for Distances Avoiding a Failed Node or Link. *SIAM Journal on Computing*, 37:1299–1318, 2008. doi:10.1137/S0097539705429847.
- 15 Ran Duan and Seth Pettie. Dual-failure Distance and Connectivity Oracles. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009. URL: <https://dl.acm.org/citation.cfm?id=1496770.1496826>.
- 16 Ran Duan and Tianyi Zhang. Improved Distance Sensitivity Oracles via Tree Partitioning. In *Proceedings of the 15th Algorithms and Data Structures Symposium (WADS)*, pages 349–360, 2017. doi:10.1007/978-3-319-62127-2_30.
- 17 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved Distance Sensitivity Oracles via Fast Single-Source Replacement Paths. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 748–757, 2012. doi:10.1109/FOCS.2012.17.
- 18 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster Replacement Paths and Distance Sensitivity Oracles. *ACM Transaction on Algorithms*, 16:15:1–15:25, 2020. doi:10.1145/3365835.
- 19 Yong Gu and Hanlin Ren. Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021. To appear.
- 20 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhao Xu. Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021. To appear.
- 21 Manoj Gupta, Rahul Jain, and Nitiksha Modi. Multiple Source Replacement Path Problem. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 339–348, 2020. doi:10.1145/3382734.3405714.
- 22 Manoj Gupta and Aditi Singh. Generic Single Edge Fault Tolerant Exact Distance Oracle. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 72:1–72:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.72.
- 23 John Hershberger and Subhash Suri. Vickrey Prices and Shortest Paths: What is an edge worth? In *Proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS)*, pages 252–259, 2001. doi:10.1109/SFCS.2001.959899.
- 24 John Hershberger and Subhash Suri. Erratum to “Vickrey Pricing and Shortest Paths: What is an edge worth?”. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS)*, page 809, 2002. doi:10.1109/SFCS.2002.1182006.

- 25 François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014. doi:10.1145/2608628.2608664.
- 26 Kavindra Malik, A. K. Mittal, and Sumit K. Gupta. The k Most Vital Arcs in the Shortest Path Problem. *Operations Research Letters*, 8:223–227, 1989. doi:10.1016/0167-6377(89)90065-5.
- 27 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A Faster Computation of the Most Vital Edge of a Shortest Path. *Information Processing Letters*, 79:81–85, 2001. doi:10.1016/S0020-0190(00)00175-7.
- 28 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the Most Vital Node of a Shortest Path. *Theoretical Computer Science*, 296:167–177, 2003. doi:10.1016/S0304-3975(02)00438-3.
- 29 Merav Parter and David Peleg. Sparse Fault-Tolerant BFS Structures. *ACM Transactions on Algorithms*, 13:11:1–11:24, 2016. doi:10.1145/2976741.
- 30 Hanlin Ren. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. In *Proceedings of the 28th European Symposium on Algorithms (ESA)*, pages 79:1–79:13, 2020. doi:10.4230/LIPIcs.ESA.2020.79.
- 31 Hanlin Ren. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. *CoRR*, abs/2007.11495, 2020. ArXiv preprint. Full version of [30]. arXiv:2007.11495.
- 32 Liam Roditty and Uri Zwick. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. *ACM Transaction on Algorithms*, 8:33:1–33:11, 2012. doi:10.1145/2344422.2344423.
- 33 Mikkel Thorup. Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. *Journal of the ACM*, 46:362–394, 1999. doi:10.1145/316542.316548.
- 34 Mikkel Thorup and Uri Zwick. Approximate Distance Oracles. *Journal of the ACM*, 52:1–24, 2005. doi:10.1145/1044731.1044732.
- 35 Virginia Vassilevska Williams. Multiplying Matrices Faster Than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. doi:10.1145/2213977.2214056.
- 36 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *Journal of the ACM*, 65:27:1–27:38, 2018. doi:10.1145/3186893.
- 37 Oren Weimann and Raphael Yuster. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Transactions on Algorithms*, 9:14:1–14:13, 2013. doi:10.1145/2438645.2438646.

Synchronized Planarity with Applications to Constrained Planarity Problems

Thomas Bläsius 

Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Germany

Simon D. Fink  

Faculty of Informatics and Mathematics, Universität Passau, Germany

Ignaz Rutter  

Faculty of Informatics and Mathematics, Universität Passau, Germany

Abstract

We introduce the problem SYNCHRONIZED PLANARITY. Roughly speaking, its input is a loop-free multi-graph together with synchronization constraints that, e.g., match pairs of vertices of equal degree by providing a bijection between their edges. SYNCHRONIZED PLANARITY then asks whether the graph admits a crossing-free embedding into the plane such that the orders of edges around synchronized vertices are consistent. We show, on the one hand, that SYNCHRONIZED PLANARITY can be solved in quadratic time, and, on the other hand, that it serves as a powerful modeling language that lets us easily formulate several constrained planarity problems as instances of SYNCHRONIZED PLANARITY. In particular, this lets us solve CLUSTERED PLANARITY in quadratic time, where the most efficient previously known algorithm has an upper bound of $O(n^8)$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Graphs and surfaces

Keywords and phrases Planarity Testing, Constrained Planarity, Cluster Planarity, Atomic Embeddability

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.19

Related Version *Full Version*: <https://arxiv.org/abs/2007.15362>

Funding Work partially supported by DFG-grant Ru-1903/3-1.

Simon D. Fink: DFG-grant Ru-1903/3-1.

Ignaz Rutter: DFG-grant Ru-1903/3-1.

1 Introduction

A graph is *planar* if it admits an embedding into the plane that has no edge crossings. Planarity is a well-studied concept that facilitates beautiful mathematical structures [20, 9], allows for more efficient algorithms [19], and serves as a cornerstone in the context of network visualization [26]. It is not surprising that various generalizations, extensions, and constrained variants of the PLANARITY problem have been studied [24]. Examples are CLUSTERED PLANARITY, where the embedding has to respect a laminar family of clusters [22, 8]; CONSTRAINED PLANARITY, where the orders of edges incident to vertices are restricted, e.g., by PQ-trees [6]; and SIMULTANEOUS PLANARITY, where two graphs sharing a common subgraph must be embedded such that their embeddings coincide on the shared part [5].

For planar embeddings, there is the important notion of *rotation*. The rotation of a vertex is the counter-clockwise cyclic order of incident edges around it. Many of the above planarity variants come down to the question whether there are embeddings of one or multiple graphs such that the rotations of certain vertices are in sync in a certain way. Inspired by this observation, by the ATOMIC EMBEDDABILITY problem [15], and by the cluster decomposition tree (CD-tree) [8], we introduce a new planarity variant. SYNCHRONIZED PLANARITY has a



© Thomas Bläsius, Simon D. Fink, and Ignaz Rutter;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



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

loop-free multi-graph together with two types of synchronization constraints as input. Each *Q-constraint* is given as a subset of vertices together with a fixed reference rotation for each of these vertices. The *Q-constraint* is satisfied if and only if either all these vertices have their reference rotation or all these vertices have the reversed reference rotation. Vertices appearing in *Q-constraints* are called *Q-vertices* and all remaining vertices are *P-vertices*.¹ A *P-constraint* between two *P-vertices* u and v defines a bijection between the edges incident to u and v . It is satisfied if and only if u and v have the opposite rotation under this bijection. We require that the *P-constraints* form a matching, that is, no vertex appears in more than one *P-constraint*. The decision problem SYNCHRONIZED PLANARITY now asks whether the given graph can be embedded such that all *Q-* and all *P-constraints* are satisfied.

SYNCHRONIZED PLANARITY serves as a powerful modeling language that lets us express various other planarity variants using simple linear-time reductions. Specifically, we provide such reductions for CLUSTERED PLANARITY, ATOMIC EMBEDDABILITY, PARTIALLY PQ-CONSTRAINED PLANARITY, and SIMULTANEOUS EMBEDDING WITH FIXED EDGES with a connected shared graph (CONNECTED SEFE). Our main contribution is an algorithm that solves SYNCHRONIZED PLANARITY, and thereby all the above problems, in quadratic time.

1.1 Technical Contribution

Our result impacts different planarity variants that have been studied previously. Before discussing them individually in the context of previous publications, we point out a common difficulty that has been a major barrier for all of them, and briefly sketch how we resolve it.

Consider the following constraint on the rotation of a single vertex. Assume its incident edges are grouped and we only allow orders where no two groups alternate, that is, if e_1, e_2 are in one group and e_3, e_4 are in a different group, then the circular subsequence e_1, e_3, e_2, e_4 and its inverse are forbidden. Such restrictions have been called *partition constraints* before [8], and they naturally emerge at cut-vertices where each incident 2-connected component forms a group. A single partition constraint is not an issue by itself, but it becomes difficult to deal with in combination with further restrictions. This is why cut-vertices and disconnected clusters are a major obstacle for SEFE [5] and CLUSTERED PLANARITY [8], respectively.

The same issues appear for SYNCHRONIZED PLANARITY, when we have a cut-vertex v that is involved in *P-constraints*, that is, its rotation has to be synchronized with the rotation of a different vertex u . We deal with these situations as follows, depending on whether u is also a cut-vertex or not. If not, it is rather well understood which embedding choices impact the rotation of u and we can propagate this from u to v .² This breaks the synchronization of u and v down into the synchronization of smaller embedding choices. This is a well-known technique that has been used before [6, 17]. If u is also a cut-vertex, we are forced to actually deal with the embedding choices emerging at cut-vertices. This is done by “encapsulating” the restrictions on the rotations of u and v that are caused by the fact that they are cut-vertices. All additional restrictions coming from embedding choices in the 2-connected components are pushed away by introducing additional *P-constraints*. After this, the cut-vertices u and v have very simple structure, which can be resolved by essentially joining them together. This procedure is formally described in Section 3.2 and illustrated in Figures 2 and 3.

¹ The names are based on PQ-trees, where *Q-* and *P-*nodes have fixed and arbitrary rotation, respectively.

² We can also do this if v is not a cut-vertex.

This solution can be seen as combinatorial perspective on the recent breakthrough result by Fulek and Tóth [15], who resolved the cut-vertex issue by applying an idea coming from Carmesin’s work [10]. While Carmesin works with 2-dimensional simplicial complexes, Fulek and Tóth achieve their result by transferring Carmesin’s idea to the setting of topological graphs on surfaces and combining it with tools from their work on thickenability. Our work transfers the problem and its solution back to an entirely combinatorial treatment of topological graphs in the plane. This further simplification allows us to more clearly highlight the key insight that makes the algorithm tick and at the same time provides access to a wide range of algorithmic tools for speeding up the computations. Due to space constraints, proofs of statements marked with a star are only given in the full version.

1.2 Related Work

CLUSTERED PLANARITY was first considered by Lengauer [22] and later rediscovered by Feng et al. [13]. In both cases, the authors give polynomial-time algorithms for the case that each cluster induces a connected graph. The complexity of the general problem that allows disconnected clusters has been open for 30 years. In that time, many special cases have been shown to be polynomial-time solvable [3, 11, 14, 16] before Fulek and Tóth [15] recently settled CLUSTERED PLANARITY in P. The core ingredient for this is their $O(n^8)$ algorithm for the ATOMIC EMBEDDABILITY problem. It has two graphs G and H as input. Roughly speaking, H describes a 3-dimensional molecule structure with atoms represented by spheres and connections (a.k.a. pipes) represented by cylinders. The other graph G comes with a map to the molecule structure that maps each vertex to an atom such that two neighboring vertices lie on the same atom or on two atoms connected by a pipe. ATOMIC EMBEDDABILITY then asks whether G can be embedded onto the molecule structure such that no edges cross.

ATOMIC EMBEDDABILITY has been introduced as a generalization of the THICKENABILITY problem that appears in computational topology [1]. It can be shown that ATOMIC EMBEDDABILITY and THICKENABILITY are linear-time equivalent [15]. Thus, the above $O(n^8)$ algorithm for ATOMIC EMBEDDABILITY also solves THICKENABILITY and SYNCHRONIZED PLANARITY. In a preprint, Carmesin [10] proves a Kuratowski-style characterization of THICKENABILITY, which he claims yields a quadratic algorithm as a byproduct. While it is believable that the running time of his algorithm is polynomial, a detailed runtime analysis is missing. In light of this, we only compare our algorithm to the $O(n^8)$ -algorithm by Fulek and Tóth. For a detailed comparison of their solution to ATOMIC EMBEDDABILITY and our solution to SYNCHRONIZED PLANARITY, we refer to the full version.

To finally solve CLUSTERED PLANARITY, Fulek and Tóth [15] use the reduction of Cortese and Patrignani [12] to INDEPENDENT FLAT CLUSTERED PLANARITY, which they then reduce further to THICKENABILITY. The last reduction to THICKENABILITY is based on a combinatorial characterization of THICKENABILITY by Neuwirth [23], which basically states that multiple graphs have to be embedded consistently, that is, such that the rotation is synchronized between certain vertex pairs of different graphs. Via the reduction from CONNECTED SEFE to CLUSTERED PLANARITY given by Angelini and Da Lozzo [2], the above result extends to CONNECTED SEFE, which was a major open problem in the context of simultaneous graph representations [7]. We flatten this chain of reductions by giving a simple linear reduction from each of the problems CONNECTED SEFE, CLUSTERED PLANARITY, and ATOMIC EMBEDDABILITY to SYNCHRONIZED PLANARITY, yielding quadratic-time algorithms for all of them. Due to space constraints, the reductions are only given in the full version. Moreover, the problem PARTIALLY PQ-CONSTRAINED PLANARITY, for which we also give a linear reduction to SYNCHRONIZED PLANARITY, has been solved in polynomial time before, but only for biconnected graphs [6] and in the non-partial setting where either all or none of the edges of a vertex are constrained [17].

2 Preliminaries

A *partition* of a base set X is a grouping of its elements into non-empty subsets, the *cells*, so that every element is contained in exactly one cell. We assume a set implementation allowing constant-time insertion and removal of elements, such as doubly-linked lists with pointers stored with the elements. When referring to graphs, we generally mean *loop-free multi-graphs*. A *(multi-)star* consists of a *center vertex* connected by multiple, possibly parallel, edges to its *ray vertices*. A *k-wheel* is a *k-cycle*, where each node is also connected to an additional central node. Furthermore, we assume a graph representation that allows efficient manipulation, such as an adjacency list with doubly-linked lists.

Drawings, Embeddings and Cyclic Orders. A *(topological) drawing* Γ of a graph is a mapping of every vertex v to a point $p_v \in \mathbb{R}^2$ in the plane and a mapping of every edge $\{u, v\}$ to a Jordan arc having p_u and p_v as endpoints. A drawing uniquely defines cyclic orders of edges incident to the same vertex. Drawings with the same cyclic orders are considered equivalent, their equivalence class is called *(combinatorial) embedding*. For an embedding \mathcal{E} , we use $\mathcal{E}(u)$ to denote the cyclic order of the edges incident to u as given by \mathcal{E} , which is also called the *rotation* of u . For a (cyclic) order $\sigma = \langle x_1, \dots, x_k \rangle$ of k elements, we use $\bar{\sigma} = \langle x_k, \dots, x_1 \rangle$ to denote its reversal.

The Synchronized Planarity Problem. An instance is a tuple $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$, where

1. $G = (P \cup Q, E)$ is a (loop-free) multi-graph with P-vertices P and Q-vertices Q ,
2. \mathcal{Q} is a partition of Q ,
3. ψ is a mapping that assigns a rotation to each Q-vertex, and
4. \mathcal{P} is a set of triples (u, v, φ_{uv}) , where u and v are P-vertices of the same degree, φ_{uv} is a bijection between their incident edges, and each P-vertex occurs at most once in \mathcal{P} .

We call the triples $\rho = (u, v, \varphi_{uv})$ in \mathcal{P} *pipes*. Pipes are not directed and we identify (u, v, φ_{uv}) and (v, u, φ_{vu}) with $\varphi_{vu} = \varphi_{uv}^{-1}$. We also define $\deg(\rho) = \deg(u) = \deg(v)$. If two P-vertices are connected by a pipe, we call them *matched*; all other P- and Q-vertices are *unmatched*.

The planar embedding \mathcal{E} of G *satisfies the cell* $X \in \mathcal{Q}$ if it is either $\mathcal{E}(v) = \psi(v)$ for all $v \in X$ or $\mathcal{E}(v) = \bar{\psi(v)}$ for all $v \in X$. We say that the embedding satisfies the *Q-constraints* if it satisfies all cells, that is, vertices in the same cell of the partition \mathcal{Q} are consistently oriented. The embedding \mathcal{E} *satisfies the pipe* $\rho = (u, v, \varphi_{uv})$ if $\varphi_{uv}(\mathcal{E}(u)) = \bar{\mathcal{E}(v)}$, that is, they have opposite rotations under the bijection φ_{uv} . We say that the embedding satisfies the *P-constraints* if it satisfies all pipes. The embedding \mathcal{E} is called *valid* if it satisfies the P-constraints and the Q-constraints. The problem SYNCHRONIZED PLANARITY asks whether a given instance $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$ admits a valid embedding.

PQ-Trees and Embedding Trees. A *PQ-tree* represents a set of circular orders of its leaves by partitioning its inner nodes into two classes: For *Q-nodes* the rotation of incident edges is fixed up to reversal, for *P-nodes*, this order can be chosen arbitrarily. Rooted PQ-trees have initially been studied by Booth and Lueker [9]. There is an equivalence between rooted and unrooted PQ-trees [21], where the latter are also called PC-trees [25]. We thus do not distinguish them and simply use the term PQ-trees. Note that a P-node with three or less neighbors allows the same permutations as a Q-node of the same degree. We thus assume P-nodes to have degree at least 4. We consider a PQ-tree *trivial* if it consists of a single inner P-node (with at least four leaves). Otherwise, it consists of a single Q-node with at least two leaves, or it contains at least two inner nodes, all of which have degree at least 3.

For a vertex of a planar biconnected graph, all rotations induced by planar embeddings can efficiently be represented by a PQ-tree [9]. This PQ-tree is also called the *embedding tree* of the respective node. In the context of SYNCHRONIZED PLANARITY, we assume that the embedding tree of a vertex does not allow rotations that would result in a Q-vertex v having any other rotation than its default ordering $\psi(v)$ or its reverse $\overline{\psi(v)}$. To ensure this, we can subdivide each edge incident to v and connect each pair of two of the new nodes if the edges they subdivide are consecutive in the cyclic order $\psi(v)$ [17]. Note that this generates a k -wheel with center v and that there are exactly two planar rotations of the center of a wheel, which are the reverse of each other. We always generate the embedding trees based on the graph where each Q-vertex in G is temporarily replaced with its respective wheel.

Connected Components. A separating k -set is a set of k vertices whose removal increases the number of connected components. Separating 1-sets are called *cut-vertices*, while separating 2-sets are called *separation pairs*. A connected graph is *biconnected* if it does not have a cut-vertex. A biconnected graph is *triconnected* if it does not have a separation pair. Maximal biconnected subgraphs are called *blocks*. A vertex that is not a cut-vertex and thus resides within an unique block is called *block-vertex*.

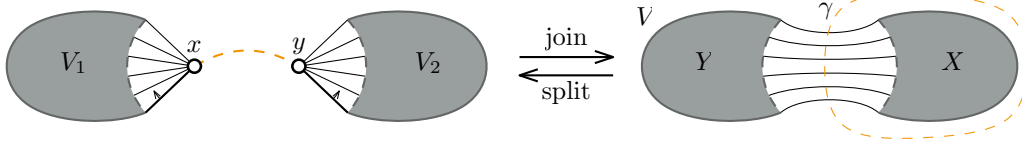
Hopcroft and Tarjan [20] define a graph decomposition into triconnected components, also called *SPQR-tree* [4], where the components come in three shapes: *bonds* consist of two *pole* vertices connected by multiple parallel edges, *polygons* consist of a simple cycle, and *rigids*, whose embeddings are unique up to reflection. Each edge of these components is either *real*, representing a single edge of the original graph, or *virtual*, representing a subgraph.

Every planar embedding of a biconnected planar graph can be obtained from an arbitrary planar embedding by flipping its rigids and reordering the parallel edges in its bonds [20]. The decomposition can be computed in linear time [18] and can be used to compute the embedding trees in linear time [6, Section 2.5].

Splits and Joins of Graphs and Embeddings. Let $G = (V, E)$ be a graph. We call a partition $C = (X, Y)$ of V into two disjoint cells a *cut* of G . The edges $E(C)$ that have their endpoints in different cells are called *cut edges*. The *split* of G at $C = (X, Y)$ is the disjoint union of the two graphs obtained by contracting X and Y to a single vertex x and y , respectively (keeping possible multi-edges); see Figure 1. Note that the edges incident to x and y are exactly the cut edges, yielding a natural bijection φ_{xy} between them. Conversely, given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ and vertices $x \in V_1, y \in V_2$ together with a bijection φ_{xy} between their incident edges, their *join* along φ_{xy} is the graph $G = (V, E)$, where $V = V_1 \cup V_2 \setminus \{x, y\}$ and E contains all edges of $E_1 \cup E_2$ that are not incident to x or y , and for each edge $e = ux$ incident to x , E contains an edge uv , where v is the endpoint of $\varphi_{xy}(e)$ distinct from y ; see Figure 1. Observe that split and join are inverse operations.

We say that a planar embedding \mathcal{E} of a graph G *respects* a cut $C = (X, Y)$ if and only if for a topological planar drawing Γ of G with embedding \mathcal{E} there exists a closed curve γ such that (i) γ separates X from Y , (ii) γ crosses each edge in $E(C)$ in exactly one point, and (iii) γ does not cross any edge in $E \setminus E(C)$; see Figure 1. We say that γ *represents* C in Γ .

If \mathcal{E} respects C , a split at C preserves \mathcal{E} as follows. Let G_1 and G_2 be the graphs resulting from splitting G at C and let $x \in V_1$ and $y \in V_2$ such that φ_{xy} identifies their incident edges. Let Γ be a topological planar drawing with embedding \mathcal{E} and let γ be a curve in Γ that represents C in Γ . We obtain planar drawings Γ_1 and Γ_2 of G_1 and G_2 by contracting to a single point the side of γ that contains V_2 and V_1 , respectively. We denote by \mathcal{E}_1 and \mathcal{E}_2 the corresponding combinatorial embeddings of G_1 and G_2 . Note that by construction for



■ **Figure 1** Joining and splitting two graphs at $x \in V_1$ and $y \in V_2$. The bijection φ_{xy} between their incident edges is shown as follows: the two bold edges at the bottom are mapped to each other. The other edges are mapped according to their order following the arrow upwards (i.e. clockwise for x and counter-clockwise for y).

each vertex of $V_1 \setminus \{x\}$ the rotations in \mathcal{E} and \mathcal{E}_1 coincide, and the same holds for vertices of $V_2 \setminus \{y\}$ in \mathcal{E} and \mathcal{E}_2 . Moreover, the rotations $\mathcal{E}_1(x)$ and $\mathcal{E}_2(y)$ are determined by the order in which the edges of $E(C)$ cross γ , and therefore they are oppositely oriented, that is, $\varphi_{xy}(\mathcal{E}_1(x)) = \overline{\mathcal{E}_2(y)}$. We call embeddings \mathcal{E}_1 and \mathcal{E}_2 with this property *compatible with φ_{xy}* .

Conversely, we can join arbitrary embeddings \mathcal{E}_1 of G_1 and \mathcal{E}_2 of G_2 that are compatible with φ_{xy} by assuming that x and y lie on the outer face, removing x and y from the embeddings, and connecting the resulting half-edges according to φ_{xy} . The result is a planar embedding \mathcal{E} where for each vertex $v \in V_i \setminus \{x, y\}$ it is $\mathcal{E}(v) = \mathcal{E}_i(v)$ for $i = 1, 2$.

► **Lemma 1 (*)**. *Let $G = (V, E)$ be a planar graph and let (X, Y) be a cut of G such that X and Y induce connected subgraphs of G . Then every planar embedding of G respects (X, Y) .*

► **Lemma 2 (*)**. *Every planar embedding of a bipartite graph $G = (A \cup B, E)$ respects (A, B) .*

3 The Synchronized Planarity Problem

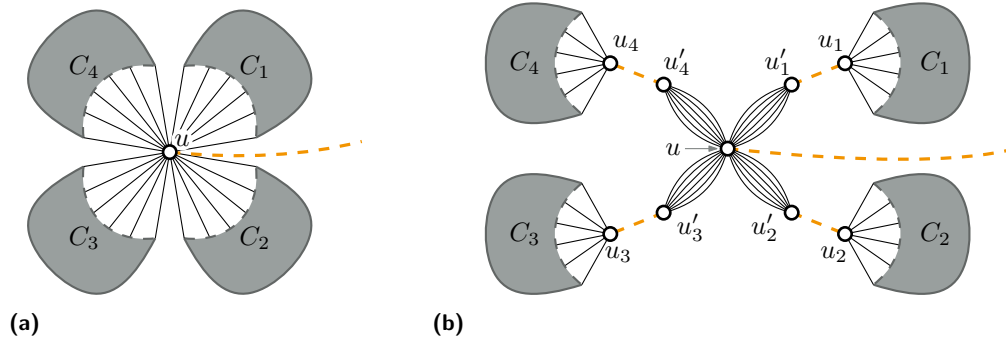
We give an algorithm for solving SYNCHRONIZED PLANARITY for graphs with n vertices and m edges in $O(m^2)$ time. Without loss of generality, we assume that G has no isolated vertices and thus $m \in \Omega(n)$. Furthermore, we assume the input graph G to be planar.

3.1 High-Level Algorithm

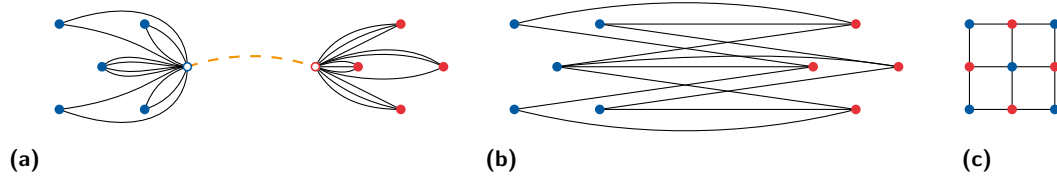
Our approach hinges on three main ingredients. The first are the three operations **EncapsulateAndJoin**, **PropagatePQ**, and **SimplifyMatching**, each of which can be applied to pipes that satisfy certain conditions. If an operation is applicable, it produces an equivalent instance I' of SYNCHRONIZED PLANARITY in linear time. Secondly we show that if none of the operations is applicable, then I has no pipes, and we give a simple linear-time algorithm for computing a valid embedding in this case. The third ingredient is a non-negative potential function ϕ for instances of SYNCHRONIZED PLANARITY. We show that it is upper-bounded by $2m$, and that each of the three operations decreases it by at least 1.

Our algorithm is therefore extremely simple; namely, while the instance still has a pipe, apply one of the operations to decrease the potential. Since the potential function is initially bounded by $2m$, at most $2m$ operations are applied, each taking $O(m)$ time. We will show that the resulting instance without pipes has size $O(m^2)$ and can be solved in linear time, thus the total running time is $O(m^2)$.

Conversion of small-degree P-vertices. The main difficulty in SYNCHRONIZED PLANARITY stems from matched P-vertices. However, P-vertices of degree up to 3 behave like Q-vertices in the sense that their rotations are unique up to reversal. Throughout this paper, we



■ **Figure 2** A matched cut-vertex (a) and the result of encapsulating it (b).



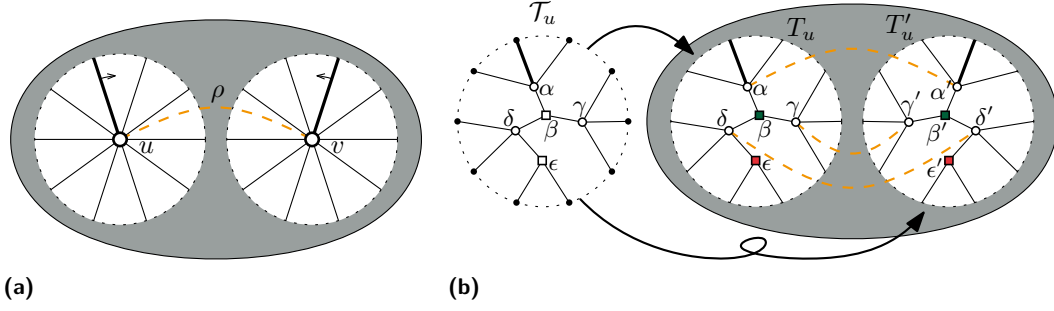
■ **Figure 3** Two (encapsulated) matched cut-vertices (a). Depending on the mapping φ , any bipartite graph can result from joining them. For example, the graph (b) can result, which is isomorphic to the square grid graph shown in (c).

implicitly assume that P-vertices of degree less than 4 are converted into Q-vertices, also converting a pipe of degree less than 4 into a Q-constraint; see the full version for additional details. We therefore assume without loss of generality that P-vertices, and in particular pipes, have degree at least 4.

3.2 The EncapsulateAndJoin Operation

The purpose of the **EncapsulateAndJoin** operation is to communicate embedding restrictions between two matched cut-vertices in two steps: First we encapsulate the cut-vertices into their own independent star components, also disconnecting their incident blocks from each other. In the second step, we join the stars. Figures 2 and 3 show an example.

For an instance $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$ of SYNCHRONIZED PLANARITY, let $\rho = (u, v, \varphi_{uv})$ be a pipe matching two cut-vertices u, v of two (not necessarily distinct) connected components C_u, C_v of G . Operation **EncapsulateAndJoin** (ρ, I) can be applied resulting in an instance $I' = (G', \mathcal{P}', \mathcal{Q}', \psi')$ using the following two steps. We first preprocess both cut-vertices to *encapsulate* them into their own separate star components. Let C_1, \dots, C_k be the connected components of $C_u - u$. We split C_u along the cuts $(V(C_i), V \setminus V(C_i))$ for $i = 1, \dots, k$. We denote the vertices resulting from the split along $(V(C_i), V \setminus V(C_i))$ as u_i and u'_i , where u_i results from contracting $V \setminus V(C_i)$ and u'_i results from contracting $V(C_i)$. Note that, after all splits, u is the center of a star C'_u whose ray vertices are the u'_i . We add the pipes $(u_i, u'_i, \varphi_{u_i u'_i})$ for $i = 1, \dots, k$; see Figure 2. The same procedure is also applied to v , resulting in an intermediate instance I^* . In the second step, we *join* the connected components C'_u and C'_v at u and v along the mapping φ_{uv} of ρ into a component C_{uv} . We also remove the pipe ρ from I^* ; all other parts of the instance remain unchanged. Figure 3 shows a possible result of joining two stars.



■ **Figure 4** A block-vertex u matched with vertex v (a); the bijection φ_{uv} maps the bold edge of u to the bold edge of v , the remaining edges are mapped according to their order, clockwise around u and counter-clockwise around v . The result of applying **PropagatePQ** (u, I) (b). Note that the second inserted tree T'_u is mirrored with respect to T_u . Q-vertices and -nodes are drawn as squares while P-vertices and -nodes are drawn as disks.

► **Lemma 3.** *Applying **EncapsulateAndJoin** to a pipe ρ yields an equivalent instance in $O(\deg(\rho))$ time.*

Proof. By Lemma 1, a valid embedding \mathcal{E} of an instance I respects each of the cuts $(V(C_i), V \setminus V(C_i))$ for $i = 1, \dots, k$, yielding a planar embedding \mathcal{E}^* of I^* . By construction, it is $\mathcal{E}^*(u_i) = \overline{\mathcal{E}^*(u'_i)}$ for $i = 1, \dots, k$, that is, each new pipe $(u_i, u'_i, \varphi_{u_i u'_i})$ is satisfied and \mathcal{E}^* is a valid embedding of I^* . Conversely, if \mathcal{E}^* is a valid embedding of I^* , we can join u_i with u'_i for $i = 1, \dots, k$ to obtain a valid planar embedding \mathcal{E} of I , as the pipe $(u_i, u'_i, \varphi_{u_i u'_i})$ ensures that \mathcal{E}^* is compatible with $\varphi_{u_i u'_i}$. The same applies to C_v .

If \mathcal{E}^* is a valid embedding for I^* , it satisfies the pipe (u, v, φ_{uv}) and we can join the embedding at u and v via φ_{uv} to obtain a planar embedding \mathcal{E}' of G' . Since the rotations of vertices different from u, v are unaffected, \mathcal{E}' is valid for I' . Conversely, assume that \mathcal{E}' is a valid embedding for I' . Note that joining two stars at their centers yields a bipartite graph consisting of the rays of the former stars. Thus C_{uv} is bipartite, and by Lemma 2 every embedding respects the cut of the bipartition. Thus, we can split \mathcal{E}' and obtain a valid embedding of I^* .

As the operation affects exactly the edges incident to u and v and potentially creates a new structure with size proportional to their number, its running time is linear in the degree of the affected pipe. ◀

Observe that this operation replaces a pipe and two cut-vertices by smaller pipes and smaller cut-vertices, respectively. Through multiple applications of **EncapsulateAndJoin** we can thus step by step decrease the degree of cut-vertex-to-cut-vertex pipes, until there are none left in the instance. Note that **EncapsulateAndJoin** can yield an arbitrary bipartite component. If the component is non-planar, we abort and report a no-instance.

3.3 The PropagatePQ Operation

The operation **PropagatePQ** communicates embedding restrictions of a biconnected component across a pipe. These restrictions are represented by the embedding tree of the matched P-vertex of interest. Both endpoints of the pipe are replaced by copies of this tree. To ensure that both copies are embedded in a compatible way, we synchronize their inner nodes using pipes and Q-constraints; see Figure 4.

For an instance $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$ of SYNCHRONIZED PLANARITY, let u be a block-vertex matched by a pipe $\rho = (u, v, \varphi_{uv})$. If the embedding tree \mathcal{T}_u of u is non-trivial (i.e., it not only consists of a single P-node), then the operation **PropagatePQ** (u, I) can be applied, resulting in an instance $I' = (G', \mathcal{P}', \mathcal{Q}', \psi')$ as follows. We turn the PQ-tree \mathcal{T}_u into a tree T_u by interpreting Q-nodes as Q-vertices and P-nodes as P-vertices. To construct G' from G , we replace u with T_u by reconnecting the incident edges of u to the respective leaves of T_u . We also replace v by a second copy T'_u of T_u by reconnecting an edge e incident to v to the leaf of T'_u that corresponds to $\varphi_{vu}(e)$. For a vertex α of T_u we denote the corresponding vertex of T'_u by α' . For an edge $\alpha\beta$ of T_u we define $\varphi_{T_u T'_u}(\alpha\beta) = \alpha'\beta'$. For each Q-vertex α of T_u , we define $\psi'(\alpha)$ according to the rotation of the corresponding Q-node in \mathcal{T}_u . For the Q-vertex α' of T'_u , we define $\psi'(\alpha') = \overline{\varphi_{T_u T'_u}(\psi'(\alpha))}$. For all other Q-vertices of I , ψ' coincides with ψ . We define the partition $\mathcal{Q}' = \mathcal{Q} \cup \{\{\alpha, \alpha'\} \mid \alpha \text{ is a Q-vertex of } T_u\}$. For each P-vertex α of T_u , we define a pipe $\rho_\alpha = (\alpha, \alpha', \varphi_{\alpha\alpha'})$ with $\varphi_{\alpha\alpha'}(e) = \varphi_{T_u T'_u}(e)$ for each edge e incident to α . Finally, we define the matching $\mathcal{P}' = (\mathcal{P} \setminus \{\rho\}) \cup \{\rho_\alpha \mid \alpha \text{ is a P-vertex of } T_u\}$.

► **Lemma 4** (*). *Applying **PropagatePQ** to a block-vertex u with a non-trivial embedding tree yields an equivalent instance. If the embedding tree \mathcal{T}_u is known, operation **PropagatePQ** runs in $O(\deg(u))$ time.*

Note that the tree T'_u inserted instead of v may not be compatible with the rotations of v . In this case, the component becomes non-planar, potentially causing the later generation of an embedding tree to fail. We can then immediately report a no-instance.

Observe that since we assume \mathcal{T}_u to be non-trivial, the degrees of all P-vertices in T_u and T'_u are strictly smaller than the degree of u . Thus, by repeatedly applying **PropagatePQ**, we eventually arrive at an equivalent instance where all matched block-vertices have a trivial embedding tree. Also note that if \mathcal{T}_u consists of a single Q-node, **PropagatePQ** effectively replaces the affected pipe by two Q-vertices in the same partition. The case where \mathcal{T}_u is trivial and thus consists of a single P-node is handled by the next operation.

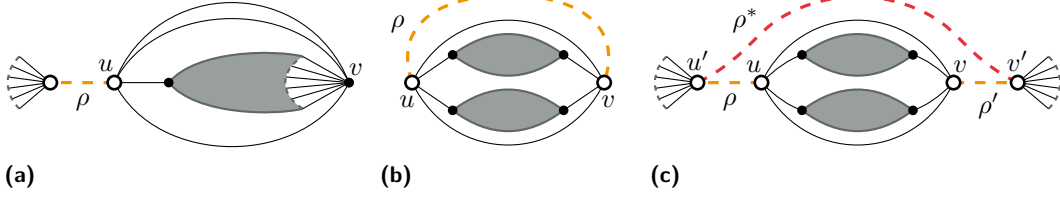
3.4 The SimplifyMatching Operation

The remaining operation is **SimplifyMatching**, which is used to resolve pipes where one side has no restrictions to be communicated to the other side. This is the case when one of the two matched vertices is a pole of a bond that allows arbitrary rotation. We distinguish three cases: i) bonds where one pole can always mimic the rotation of the other, ii) bonds where the pipe synchronizes one pole with the other (similar to the toroidal instances of Fulek and Tóth [15]), and iii) bonds that link two distinct pipes.

For an instance $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$ of SYNCHRONIZED PLANARITY, let u be a block-vertex of G whose embedding tree is trivial and that is matched by a pipe ρ . Then, its embedding is determined by exactly one triconnected component μ , which is a bond.³ Thus u is the pole of bond μ , and we call the vertex v that is the other pole of μ the *partner* of u . If v is either unmatched or a block-vertex with a trivial embedding tree, the operation **SimplifyMatching** (u, I) can be applied, resulting in an instance $I' = (G', \mathcal{P}', \mathcal{Q}', \psi')$ as follows. Note that, due to the temporary replacement of Q-vertices by wheels when computing the embedding trees, v cannot be a Q-vertex, as that would make the PQ-tree of u contain a Q-node.

- (i) If v is an unmatched P-vertex (Figure 5a), I' is obtained from I by removing ρ .

³ as a second bond would cause another P-node in the embedding tree, a rigid would cause a Q-node and polygons do not affect the embedding trees [6, Section 2.5]



■ **Figure 5** The three cases of the `SimplifyMatching` operation. In Case i (a) and Case ii (b), the pipe ρ is removed. In Case iii (c) the pipes ρ, ρ' are replaced by pipe ρ^* .

- (ii) If ρ matches u with v , it connects the two poles of the bond μ (Figure 5b). Note that the embedding trees of u and v both contain a P-node of the same degree representing μ and the pipe now requires both u and v to have the same degree. Thus, as u has a trivial embedding tree, v also has a trivial embedding tree. The rotation of the vertices is thus exclusively determined by the embedding of the bond and there are bijections δ_u and δ_v between the edges incident to u and v , respectively, and the virtual edges within the bond. We now check that these bijections are compatible with the bijection φ_{uv} given by the pipe. Let $\delta_{vu} = \delta_u^{-1} \circ \delta_v$ be a bijection between the edges incident to v and the edges incident to u , and let $\pi = \varphi_{uv} \circ \delta_{vu}$ be a permutation of the edges incident to v . If all cycles of π have the same length, I' is obtained from I by removing ρ .⁴ Otherwise, I is an invalid instance and we set I' to a trivial no-instance.
- (iii) If v is matched with a P-vertex $v' \neq u$ via pipe $\rho' = (v, v', \varphi_{vv'})$, let u' be the other endpoint of $\rho = (u, u', \varphi_{uu'})$. We remove ρ and ρ' and add the new pipe $\rho^* = (u', v', \varphi_{u'v'})$ with $\varphi_{u'v'} = \varphi_{vv'} \circ \delta_{uv} \circ \varphi_{u'u}$; see Figure 5c.

► **Lemma 5 (*)**. *Applying `SimplifyMatching` to a block-vertex u with a trivial embedding tree yields an equivalent instance in $O(\deg(u))$ time.*

3.5 Reduced and Pipe-Free Instances

With our exposition of the fundamental operations complete, we now study how to solve instances where none of those operations can be applied. We call such instances *reduced*.

► **Lemma 6**. *An instance is reduced if and only if it contains no pipes.*

Proof. Obviously, a pipe-free instance is reduced. Conversely, consider a reduced instance I . Assume, for the sake of contradiction, that I contains a pipe. We now show that this implies that one of the operations is applicable, that is, I is not reduced.

Assume that I contains no matched cut-vertices and thus all matched vertices are block-vertices. If there is a matched P-vertex with a non-trivial embedding tree, `PropagatePQ` can be applied. Otherwise, all matched P-vertices are block-vertices with trivial embedding trees and `SimplifyMatching` can be applied.

Now let u be a matched cut-vertex of maximum degree that is matched to a vertex v by a pipe ρ . If v is also a cut-vertex, we can apply `EncapsulateAndJoin`. If v is a block-vertex with a non-trivial embedding tree, we can apply `PropagatePQ`. Therefore, v must be a block-vertex with a trivial embedding tree. Now we can apply `SimplifyMatching`, unless the partner pole v' of v is a matched cut-vertex. This is however excluded,

⁴ If all cycles of π have the same length, π is *order preserving* and it is $\pi(O) = O$ for any sequence O . See [6, Lemma 2.2] or the proof to the following Lemma 5 in the full version for more details.

since $\deg(u) = \deg(v) < \deg(v')$, contradicting the maximality of $\deg(u)$. The last inequality follows from the fact that $\deg(v) \leq \deg(v')$ already holds in the block of G that contains v and v' , but as v' is a cut-vertex, it has at least one neighbor outside that block. ◀

To solve instances without pipes in linear time, note that a planar embedding of such an instance is valid if and only if it satisfies the Q-constraints. As Q-vertices only have a binary choice for their rotation, it is relatively easy to synchronize them via a 2-SAT formula. Linear-time algorithms follow from, e.g., [6], and can also be obtained from techniques similar to those used by Fulek and Tóth [15] for cubic graphs. For the sake of completeness, we present a self-contained solution in the full version.

► **Lemma 7 (*)**. *An instance of SYNCHRONIZED PLANARITY without pipes can be solved in $O(m)$ time. A valid embedding can be computed in the same time, if it exists.*

3.6 Finding a Reduced Instance

As mentioned above, we exhaustively apply the operations **EncapsulateAndJoin**, **PropagatePQ**, and **SimplifyMatching**. We claim that this algorithm terminates and yields a reduced instance after a polynomial number of steps. The key idea is that the operations always make progress by either reducing the number of pipes, or by splitting pipes into pipes of smaller degree. This suggests that, eventually, we arrive at an instance without pipes. However, there are two caveats. First, the encapsulation in the first step of **EncapsulateAndJoin** creates new pipes and thus has the potential to undo progress. Second, the smaller pipes resulting from splitting a pipe with **PropagatePQ** might cause further growth of the instance, potentially causing a super-polynomial number of steps.

We resolve both issues by using a more fine-grained measure of progress in the form of a potential function. To overcome the first issue, we show that for each application of **EncapsulateAndJoin**, the progress that is undone in the first step is outweighed by the progress made through the following join in the second step. Similarly, for the second issue, we show that the sum of the parts is no bigger than the whole when splitting pipes.

As P-vertices of degree 3 or less are converted to Q-vertices (see Section 3.1), we use $\deg^*(u) = \deg^*(v) = \deg^*(\rho) = \max\{\deg(x) - 3, 0\}$ to denote the number of incident edges that keep a P-vertex u (and also the other endpoint v of its pipe $\rho = (u, v, \varphi_{uv})$) from becoming converted to a Q-vertex. We also partition the set of all pipes \mathcal{P} into the two cells \mathcal{P}_{CC} and $\mathcal{P}_B = \mathcal{P} \setminus \mathcal{P}_{CC}$, where \mathcal{P}_{CC} contains all pipes where both endpoints are cut-vertices. We define the *potential* of an instance I as $\Phi(I) = \sum_{\rho \in \mathcal{P}_B} \deg^*(\rho) + \sum_{\rho \in \mathcal{P}_{CC}} (2\deg^*(\rho) - 1)$. We show that the operations always decrease this potential.

► **Lemma 8 (*)**. *For an instance $I = (G, \mathcal{P}, \mathcal{Q}, \psi)$ of SYNCHRONIZED PLANARITY and an instance $I' = (G', \mathcal{P}', \mathcal{Q}', \psi')$ that results from application of either **EncapsulateAndJoin**, **PropagatePQ** or **SimplifyMatching** to I , the following three properties hold:*

- (i) *The potential reduction $\Delta\Phi = \Phi(I) - \Phi(I')$ is at least 1.*
- (ii) *The number of nodes added to the graph satisfies $\Delta V = |V(G')| - |V(G)| \leq 2 \cdot \Delta\Phi + 12$.*
- (iii) *If the operation replaces a connected component C by one or multiple connected components, then each such component C' satisfies $\Delta E(C) = |E(C')| - |E(C)| \leq 2 \cdot \Delta\Phi$.*

Proof Sketch. We now analyze the effects of **EncapsulateAndJoin** on these three measures. The operations **PropagatePQ** and **SimplifyMatching** are discussed in the full version.

Operation **EncapsulateAndJoin** (ρ, I) in the first step encapsulates both cut-vertices u, v to their own star components. For each block incident to u , this introduces two new vertices that are connected by a new pipe. Let d_1, \dots, d_k be the degrees of the $k \geq 2$ ray vertices of

u after the encapsulation. As one end of the added pipes is a block-vertex, the potential is increased by $\sum_{i=1}^k \max\{d_i - 3, 0\}$. Likewise, the pipes of the k' rays with degrees $d'_1, \dots, d'_{k'}$ around v increase the potential by $\sum_{i=1}^{k'} \max\{d'_i - 3, 0\}$. Using $\deg(\rho) = \sum_{i=1}^k d_i = \sum_{i=1}^{k'} d'_i = D \geq 3$ it is $\deg^*(\rho) = \max\{D - 3, 0\} = D - 3$. In the second step, removing ρ connecting two cut-vertices together with its endpoints reduces the potential by $2\deg^*(\rho) - 1$ and we thus get $\Delta\Phi = 2 \cdot (D - 3) - 1 - \sum_{i=1}^k \max\{d_i - 3, 0\} - \sum_{i=1}^{k'} \max\{d'_i - 3, 0\}$. In the full version, we show that this yields $\Delta\Phi \geq 1$ as claimed by (i). As the encapsulation generates two vertices for each ray and the join removes two vertices, we have $\Delta V = 2k + 2k' - 2$. We also show that claim (ii) holds as $\Delta V \leq 2 \cdot (\Delta\Phi + 6)$. In the first step of **EncapsulateAndJoin**, two new components with $\deg(u) = \deg(v)$ edges each are added, which are then combined in the second step, yielding a new component with $\sum_{i=1}^k d_i$ edges. This is no larger than the components of u or v as required for (iii). ◀

With this lemma, we know that each step decreases the potential by at least 1 without growing the graph too much. As each vertex contributes at most twice its degree, initially $\Phi(I) \leq 2m$. This can then be used to bound the size of instances resulting from applying multiple operations consecutively and finally to bound the time required to find a solution for an instance.

► **Theorem 9 (*)**. *SYNCHRONIZED PLANARITY can be solved in $O(m^2)$ time.*

Proof Sketch. By Lemma 8 the potential function decreases with each applied operation. As initially $\Phi(I) \leq 2m$, a reduced instance I' is reached after $k \leq 2m$ operations. It can be shown that each connected component of I' has $O(m)$ edges, allowing an embedding tree to be computed in $O(m)$ time. Each of the k operations runs in linear time once the PQ-tree it requires is available. In total, it thus takes $O(m^2)$ time to reach a reduced instance. As its size is also in $O(m^2)$, Lemma 7 can be applied to find a solution in $O(m^2)$ time. ◀

4 Conclusion

We have given a quadratic-time algorithm for SYNCHRONIZED PLANARITY, which improves the previous $O(n^8)$ -time algorithm for the linear-time equivalent problem ATOMIC EMBEDDABILITY [15]. Similar to Goldberg and Tarjan's push-relabel algorithm, it relies on few and simple operations that can be applied pretty much in an arbitrary order. This highlights where and how progress is made and thereby clearly exposes key ideas that also underlie the algorithm for ATOMIC EMBEDDABILITY. For a direct comparison of the approaches, we refer to the full version.

The applications of SYNCHRONIZED PLANARITY include solving CLUSTERED PLANARITY, ATOMIC EMBEDDABILITY, CONNECTED SEFE and PARTIALLY PQ-CONSTRAINED PLANARITY in quadratic time, thanks to linear-time reductions to SYNCHRONIZED PLANARITY for all of them described in the full version. This improves over the previously fastest algorithms via the $O(n^8)$ -time algorithm for ATOMIC EMBEDDABILITY. In the case of CONNECTED SEFE the reduction used in [15] includes a quadratic blowup and therefore yields an $O(n^{16})$ -algorithm. Our direct linear-time reduction leads to a quadratic algorithm.

References

- 1 Hugo A. Akitaya, Radoslav Fulek, and Csaba D. Tóth. Recognizing weak embeddings of graphs. *ACM Transactions on Algorithms*, 15(4):1–27, 2019. doi:10.1145/3344549.
- 2 Patrizio Angelini and Giordano Da Lozzo. SEFE = c-planarity? *The Computer Journal*, 59(12):1831–1838, 2016. doi:10.1093/comjnl/bxw035.

- 3 Patrizio Angelini and Giordano Da Lozzo. Clustered planarity with pipes. *Algorithmica*, 81(6):2484–2526, 2019. doi:10.1007/s00453-018-00541-w.
- 4 Giuseppe Di Battista and Roberto Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. doi:10.1007/bf01961541.
- 5 Thomas Bläsius, Annette Karrer, and Ignaz Rutter. Simultaneous embedding: Edge orderings, relative positions, cutvertices. *Algorithmica*, 80(4):1214–1277, 2017. doi:10.1007/s00453-017-0301-9.
- 6 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Trans. Algorithms*, 12(2):16:1–16:46, 2016. doi:10.1145/2738054.
- 7 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 11, pages 349–381. CRC Press, Taylor & Francis Group, 2013. arXiv:1204.5853.
- 8 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theoretical Computer Science*, 609:306–315, 2016. doi:10.1016/j.tcs.2015.10.011.
- 9 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.
- 10 Johannes Carmesin. Embedding simply connected 2-complexes in 3-space – V. A refined Kuratowski-type characterisation. *CoRR*, 2017. arXiv:1709.04659v3.
- 11 Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. C-planarity of c-connected clustered graphs. *Journal of Graph Algorithms and Applications*, 12(2):225–262, 2008. doi:10.7155/jgaa.00165.
- 12 Pier Francesco Cortese and Maurizio Patrignani. Clustered planarity = flat clustered planarity. In Therese C. Biedl and Andreas Kerren, editors, *Proceedings of the 26th International Symposium on Graph Drawing and Network Visualization (GD’18)*, volume 11282 of *LNCS*, pages 23–38. Springer, 2018. doi:10.1007/978-3-030-04414-5_2.
- 13 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In Paul G. Spirakis, editor, *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA’95)*, volume 979 of *LNCS*, pages 213–226. Springer, 1995. doi:10.1007/3-540-60313-1_145.
- 14 Radoslav Fulek, Jan Kynčl, Igor Malinović, and Dömötör Pálvölgyi. Clustered planarity testing revisited. *The Electronic Journal of Combinatorics*, 22(4), 2015. doi:10.37236/5002.
- 15 Radoslav Fulek and Csaba D. Tóth. Atomic embeddability, clustered planarity, and thickenability. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’20)*, pages 2876–2895. SIAM, 2020. doi:10.1137/1.9781611975994.175.
- 16 Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel, Merijam Percan, and René Weiskircher. Advances in c-planarity testing of clustered graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD’02)*, volume 2528 of *LNCS*, pages 220–235. Springer, 2002. doi:10.1007/3-540-36151-0_21.
- 17 Carsten Gutwenger, Karsten Klein, and Petra Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *Journal of Graph Algorithms and Applications*, 12(1):73–95, 2008. doi:10.7155/jgaa.00160.
- 18 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD’00)*, volume 1984 of *LNCS*, pages 77–90. Springer, 2000. doi:10.1007/3-540-44541-2_8.
- 19 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, 4(3):221–225, 1975. doi:10.1137/0204019.
- 20 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

- 21 Wen-Lian Hsu. PC-trees vs. PQ-trees. In Jie Wang, editor, *Proceedings of the 7th Annual International Conference on Computing and Combinatorics (COCOON'01)*, volume 2108 of *LNCS*, pages 207–217. Springer, 2001. doi:10.1007/3-540-44679-6_23.
- 22 Thomas Lengauer. Hierarchical planarity testing algorithms. *Journal of the ACM*, 36(3):474–509, 1989. doi:10.1145/65950.65952.
- 23 L. Neuwirth. An algorithm for the construction of 3-manifolds from 2-complexes. *Mathematical Proceedings of the Cambridge Philosophical Society*, 64(3):603–614, 1968. doi:10.1017/S0305004100043279.
- 24 Marcus Schaefer. Toward a theory of planarity: Hanani-tutte and planarity variants. *Journal of Graph Algorithms and Applications*, 17(4):367–440, 2013. doi:10.7155/jgaa.00298.
- 25 Wei-Kuan Shih and Wen-Lian Hsu. A new planarity test. *Theoretical Computer Science*, 223(1-2):179–191, 1999. doi:10.1016/s0304-3975(98)00120-0.
- 26 Roberto Tamassia, editor. *Handbook of Graph Drawing and Visualization*. CRC Press, Taylor & Francis Group, 2014. doi:10.1201/b15385.

Efficiently Approximating Vertex Cover on Scale-Free Networks with Underlying Hyperbolic Geometry

Thomas Bläsius 

Karlsruhe Institute of Technology, Germany

Tobias Friedrich  

Hasso Plattner Institute, University of Potsdam, Germany

Maximilian Katzmann  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Finding a minimum vertex cover in a network is a fundamental NP-complete graph problem. One way to deal with its computational hardness, is to trade the qualitative performance of an algorithm (allowing non-optimal outputs) for an improved running time. For the vertex cover problem, there is a gap between theory and practice when it comes to understanding this tradeoff. On the one hand, it is known that it is NP-hard to approximate a minimum vertex cover within a factor of $\sqrt{2}$. On the other hand, a simple greedy algorithm yields close to optimal approximations in practice.

A promising approach towards understanding this discrepancy is to recognize the differences between theoretical worst-case instances and real-world networks. Following this direction, we close the gap between theory and practice by providing an algorithm that efficiently computes nearly optimal vertex cover approximations on hyperbolic random graphs; a network model that closely resembles real-world networks in terms of degree distribution, clustering, and the small-world property. More precisely, our algorithm computes a $(1 + o(1))$ -approximation, asymptotically almost surely, and has a running time of $\mathcal{O}(m \log(n))$.

The proposed algorithm is an adaption of the successful greedy approach, enhanced with a procedure that improves on parts of the graph where greedy is not optimal. This makes it possible to introduce a parameter that can be used to tune the tradeoff between approximation performance and running time. Our empirical evaluation on real-world networks shows that this allows for improving over the near-optimal results of the greedy approach.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Random network models; Mathematics of computing → Approximation algorithms; Mathematics of computing → Random graphs

Keywords and phrases vertex cover, approximation, random graphs, hyperbolic geometry, efficient algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.20

Related Version *Full Version:* <https://arxiv.org/abs/2010.02787> [7]

Funding This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Grant No. 390859508.

1 Introduction

A *vertex cover* of a graph is a subset of the vertices that leaves the graph edgeless upon deletion. Since the problem of finding a smallest vertex cover is NP-complete [19], there are probably no algorithms that solve it efficiently. Nevertheless, the problem is highly relevant due to its applications in computational biology [1], scheduling [13], and internet security [14]. Therefore, there is an ongoing effort in exploring methods that can be used in practice [2, 3], and while they often work well, they still cannot guarantee efficient running times.



© Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 20; pp. 20:1–20:15

Leibniz International Proceedings in Informatics



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

A commonly used approach to overcoming this issue are approximation algorithms. There, the idea is to settle for a near-optimal solution while guaranteeing an efficient running time. For the vertex cover problem, a simple greedy approach computes an approximation in linear time by iteratively adding the vertex with the largest degree to the cover and removing it from the graph. In general graphs, this algorithm, which we call *standard greedy*, cannot guarantee a better approximation ratio than $\log(n)$, i.e., there are graphs where it produces a vertex cover whose size exceeds the one of an optimum by a factor of $\log(n)$. This can be improved to a 2-approximation using a simple linear-time algorithm. The best known polynomial time approximation reduces the factor to $2 - \Theta(\log(n)^{-1/2})$ [18]. However, there is reason to believe that it is NP-hard to approximate an optimal vertex cover within a factor of $2 - \varepsilon$ for all $\varepsilon > 0$ [20] and it is proven that finding a $\sqrt{2}$ -approximation is NP-hard [26].

Therefore, it is very surprising that the standard greedy algorithm not only beats the 2-approximation on autonomous systems graphs like the internet [25], it also performs well on many real-world networks, obtaining approximation ratios that are very close to 1 [11]. So the theoretical bounds do not match what is observed in practice. One approach to explaining this discrepancy is to consider the differences between the examined instances. Theoretical bounds are often obtained by designing worst-case instances. However, real-world networks rarely resemble the worst case. More realistic statements can be obtained by making assumptions about the solution space [4, 9], or by restricting the analysis to networks with properties that are observed in the real world.

Many real networks, like social networks, communication networks, or protein-interaction networks, are considered to be *scale-free*. Such graphs feature a power-law degree distribution (only few vertices have high degree, while many vertices have low degree), high clustering (two vertices are likely to be adjacent if they have a common neighbor), and a small diameter.

Previous efforts to obtain more realistic insights into the approximability of the vertex cover problem have focused on networks that feature only one of these properties, namely a power-law degree distribution [10, 16, 28]. With this approach, guarantees for the approximation factor of the standard greedy algorithm were improved to a constant, compared to $\log(n)$ on general graphs [10]. Moreover, it was shown that it is possible to compute an expected $(2 - \varepsilon)$ -approximation for a constant ε , in polynomial time on such networks [16] and this was later improved to about 1.7 depending on properties of the distribution [28]. However, it was also shown that even on graphs that have a power-law degree distribution, the vertex cover problem remains NP-hard to approximate within some constant factor [10]. This indicates, that focusing on networks that only feature a power-law degree distribution, is not sufficient to explain why vertex cover can be approximated so well in practice.

The goal of this paper is to close this gap between theory and practice, by considering a random graph model that features all of the three mentioned properties of scale-free networks. The *hyperbolic random graph model* was introduced by Krioukov et al. [21] and it was shown that the graphs generated by the model have a power-law degree distribution and high clustering [17], as well as a small diameter [24]. Consequently, they are good representations of many real-world networks [8, 15, 27]. Additionally, the model is conceptually simple, making it accessible to mathematical analysis. Therefore, it has proven to be a useful framework to theoretically explain why algorithms work well in practice [6]. In fact, it has been shown that the vertex cover problem can be solved exactly in polynomial time on hyperbolic random graphs, with high probability [5]. However, we note that the degree of the polynomial is unknown and on large networks even quadratic algorithms are not efficient enough to obtain results in a reasonable amount of time.

In this paper, we link the success of the standard greedy approach to structural properties of hyperbolic random graphs, identify the parts of the graph where it does not behave optimally, and use these insights to derive a new approximation algorithm. On hyperbolic random graphs, this algorithm achieves an approximation ratio of $1 + o(1)$, asymptotically almost surely, and maintains an efficient running time of $\mathcal{O}(m \log(n))$, where n and m denote the number of vertices and edges in the graph, respectively. Since the average degree of hyperbolic random graphs is constant [17], this implies a quasi-linear running time on such networks. Moreover, we introduce a parameter that can be used to tune the tradeoff between approximation quality and running time of the algorithm, facilitating an improvement over the standard greedy approach. While our algorithm depends on the coordinates of the vertices in the hyperbolic plane, we propose an adaption of it that is oblivious to the underlying geometry and compare its approximation performance to the standard greedy algorithm on a selection of real-world networks. On average our algorithm reduces the error of the standard greedy approach to less than 50%. The evaluation of our experiments can be found in the full version of the paper [7].

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. We denote the number of vertices and edges in G with n and m , respectively. The number of vertices in a set $S \subseteq V$ is denoted by $|S|$. The *neighborhood* of a vertex v is defined as $N(v) = \{w \in V \mid \{v, w\} \in E\}$. The size of the neighborhood, called the *degree* of v , is denoted by $\deg(v) = |N(v)|$. For a subset $S \subseteq V$, we use $G[S]$ to denote the *induced subgraph* of G obtained by removing all vertices in $V \setminus S$.

The Hyperbolic Plane. After choosing a designated origin O in the two-dimensional hyperbolic plane, together with a reference ray starting at O , a point p is uniquely identified by its *radius* $r(p)$, denoting the hyperbolic distance to O , and its *angle* (or *angular coordinate*) $\varphi(p)$, denoting the angular distance between the reference ray and the line through p and O . The hyperbolic distance between two points p and q is given by

$$\text{dist}(p, q) = \text{acosh}(\cosh(r(p)) \cosh(r(q)) - \sinh(r(p)) \sinh(r(q)) \cos(\Delta_\varphi(p, q))),$$

where $\cosh(x) = (e^x + e^{-x})/2$, $\sinh(x) = (e^x - e^{-x})/2$ (both growing as $e^x/2 \pm o(1)$), and $\Delta_\varphi(p, q) = \pi - |\pi - |\varphi(p) - \varphi(q)||$ denotes the angular distance between p and q . If not stated otherwise, we assume that computations on angles are performed modulo 2π .

In the hyperbolic plane a disk of radius r has an area of $2\pi(\cosh(r) - 1)$ and circumference $2\pi \sinh(r)$. Thus, the area and the circumference of such a disk grow exponentially with its radius. In contrast, this growth is polynomial in Euclidean space.

Hyperbolic Random Graphs. Hyperbolic random graphs are obtained by distributing n points independently and uniformly at random within a disk of radius R and connecting any two of them if and only if their hyperbolic distance is at most R . See Figure 1 (left) for an example. The disk radius R (which matches the connection threshold) is given by $R = 2 \log(n) + C$, where the constant $C \in \mathbb{R}$ depends on the average degree of the network, as well as the power-law exponent $\beta = 2\alpha + 1$ (for $\alpha \in (1/2, 1)$). The coordinates of the vertices are drawn as follows. For vertex v the angular coordinate, denoted by $\varphi(v)$, is drawn uniformly at random from $[0, 2\pi)$ and the radius of v , denoted by $r(v)$, is sampled according

to the probability density function $\alpha \sinh(\alpha r)/(\cosh(\alpha R) - 1)$ for $r \in [0, R]$. Thus,

$$f(r, \varphi) = \frac{1}{2\pi} \frac{\alpha \sinh(\alpha r)}{\cosh(\alpha R) - 1} = \frac{\alpha}{2\pi} e^{-\alpha(R-r)} (1 + \Theta(e^{-\alpha R} - e^{-2\alpha r})) \quad (1)$$

is their joint distribution function for $r \in [0, R]$. For $r > R$, $f(r, \varphi) = 0$.

We denote areas in the hyperbolic disk with calligraphic capital letters. The set of vertices in an area \mathcal{A} is denoted by $V(\mathcal{A})$. The probability for a given vertex to lie in \mathcal{A} is given by its measure $\mu(\mathcal{A}) = \int_{\mathcal{A}} f(r, \varphi) d\varphi dr$. The hyperbolic distance between two vertices u and v increases with increasing angular distance between them. The maximum angular distance such that they are still connected by an edge is bounded by [17, Lemma 6]

$$\theta(r(u), r(v)) = 2e^{(R-r(u)-r(v))/2} (1 + \Theta(e^{R-r(u)-r(v)})). \quad (2)$$

Hyperbolic Random Graphs with an Expected Number of Vertices. While the positions of the vertices in a hyperbolic random graph are sampled independently of each other, stochastic dependencies are introduced once the positions of some vertices are known. For example, if all vertices lie in an area \mathcal{A} , the probability for a vertex to lie outside of \mathcal{A} is 0. When these dependencies are hard to deal with (which we mention explicitly), we resort to a slightly different hyperbolic random graph model, where the positions are sampled using an inhomogeneous Poisson point process. The result of this process is a *hyperbolic random graph with n vertices in expectation*. There, the number of vertices in disjoint areas are independent random variables. Probabilistic statements for this model can then be translated back to the original hyperbolic random graph model with a small penalty in certainty. A detailed explanation can be found in the full version of the paper [7].

Probabilities. Since we are analyzing a random graph model, our results are of probabilistic nature. To obtain meaningful statements, we show that they hold *with high probability* (with probability $1 - \mathcal{O}(n^{-1})$), or *asymptotically almost surely* (with probability $1 - o(1)$). The following Chernoff bound can be used to show that certain events occur with high probability.

► **Theorem 1** (Chernoff Bound [12, Theorem 1.1]). *Let X_1, \dots, X_n be independent random variables with $X_i \in \{0, 1\}$ and let X be their sum. Then, for $\varepsilon \in (0, 1)$*

$$\Pr[X \geq (1 + \varepsilon)\mathbb{E}[X]] \leq e^{-\varepsilon^2/3 \cdot \mathbb{E}[X]}.$$

Usually, it suffices to show that a random variable does not exceed an upper bound. The following corollary shows that a bound on the expected value suffices to obtain concentration.

► **Corollary 2.** *Let X_1, \dots, X_n be independent random variables with $X_i \in \{0, 1\}$, let X be their sum, and let $f(n)$ be an upper bound on $\mathbb{E}[X]$. Then, for $\varepsilon \in (0, 1)$*

$$\Pr[X \geq (1 + \varepsilon)f(n)] \leq e^{-\varepsilon^2/3 \cdot f(n)}.$$

Moreover, the following lemma can often be used to simplify error terms.

► **Lemma 3** ([22, Lemma 2.3]). *Let $x \in \mathbb{R}$ with $x = \pm o(1)$. Then, $1/(1 + x) = 1 - \Theta(x)$.*

3 An Improved Greedy Algorithm

Previous insights about solving the vertex cover problem on hyperbolic random graphs are based on the fact that the *dominance reduction rule* reduces the graph to a remainder of simple structure [5]. This rule states that a vertex u can be safely added to the vertex cover (and, thus, be removed from the graph) if it *dominates* at least one other vertex, i.e., if there exists a neighbor $v \in N(u)$ such that all neighbors of v are also neighbors of u .

On hyperbolic random graphs, vertices near the center of the disk dominate with high probability [5, Lemma 5]. Therefore, it is not surprising that the standard greedy algorithm that computes a vertex cover by repeatedly taking the vertex with the largest degree achieves good approximation rates on such networks: Since high degree vertices are near the disk center, the algorithm essentially favors vertices that are likely to dominate and can be safely added to the vertex cover anyway.

On the other hand, after (safely) removing high-degree vertices, the remaining vertices all have similar (small) degree, meaning the standard greedy algorithm basically picks the vertices at random. Thus, in order to improve the approximation performance of the algorithm, one has to improve on the parts of the graph that contain the low-degree vertices. Based on this insight, we derive a new greedy algorithm that achieves close to optimal approximation rates efficiently. More formally, we prove the following main theorem.

► **Theorem 4.** *Let G be a hyperbolic random graph on n vertices. Given the radii of the vertices, an approximate vertex cover of G can be computed in time $\mathcal{O}(m \log(n))$, such that the approximation ratio is $(1 + o(1))$ asymptotically almost surely.*

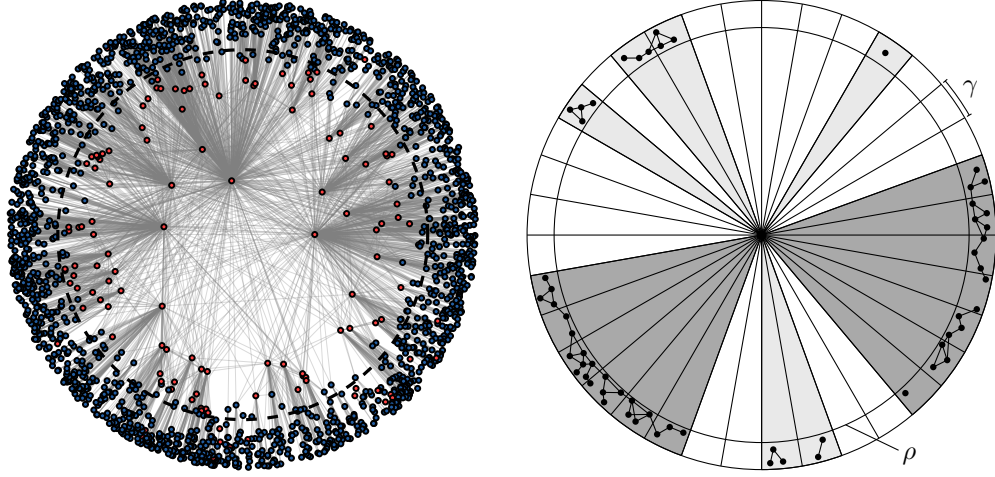
Consider the following greedy algorithm that computes an approximation of a minimum vertex cover on hyperbolic random graphs. We iterate the vertices in order of increasing radius. Each encountered vertex v is added to the cover and removed from the graph. After each step, we then identify the connected components of size at most $\tau \log \log(n)$ in the remainder of the graph, solve them optimally, and remove them from the graph as well. The constant $\tau > 0$ can be used to adjust the tradeoff between quality and running time.

This algorithm determines the order in which the vertices are processed based on their radii, which are not known for real-world networks. However, in hyperbolic random graphs, there is a strong correlation between the radius of a vertex and its degree [17]. Therefore, we can mimic the considered greedy strategy by removing vertices with decreasing degree instead. Then, the above algorithm represents an adaption of the standard greedy algorithm: Instead of greedily adding vertices with decreasing degree until all remaining vertices are isolated, we increase the quality of the approximation by solving small components exactly.

4 Approximation Performance

To analyze the performance of the above algorithm, we utilize structural properties of hyperbolic random graphs. While the power-law degree distribution and high clustering are modelled explicitly using the underlying geometry, other properties of the model, like the logarithmic diameter, emerge as a natural consequence of the first two. Our analysis is based on another emerging property: Hyperbolic random graphs decompose into small components when removing high-degree vertices.

More formally, we proceed as follows. We compute the size of the vertex cover obtained using the above algorithm, by partitioning the vertices of the graph into two sets: V_{Greedy} and V_{Exact} , denoting the vertices that were added greedily and the ones contained in small separated components that were solved exactly, respectively (see Figure 1 (left)). Clearly, we



■ **Figure 1 (Left)** A hyperbolic random graph with 1942 vertices, average degree 7.7, and power-law exponent 2.6. The vertex sets V_{Greedy} and V_{Exact} are shown in red and blue, respectively. The dashed line shows a possible threshold radius ρ . **(Right)** The disk is divided into sectors of equal width γ . Consecutive non-empty sectors form a run (grey). Narrow runs (light grey) consist of few sectors. Wide runs (dark grey) consist of many sectors. Each dark grey sector is a widening sector.

obtain a valid vertex cover for the whole graph, if we take all vertices in V_{Greedy} together with a vertex cover C_{Exact} of $G[V_{\text{Exact}}]$. Then, the approximation ratio is given by the quotient $\delta = (|V_{\text{Greedy}}| + |C_{\text{Exact}}|)/|C_{\text{OPT}}|$, where C_{OPT} denotes an optimal solution. Since all components in $G[V_{\text{Exact}}]$ are solved optimally and since any minimum vertex cover for the whole graph induces a vertex cover on $G[V']$ for any vertex subset $V' \subseteq V$, it holds that $|C_{\text{Exact}}| \leq |C_{\text{OPT}}|$. Consequently, it suffices to show that $|V_{\text{Greedy}}| \in o(|C_{\text{OPT}}|)$ in order to obtain the claimed approximation factor of $1 + o(1)$.

To bound the size of V_{Greedy} , we identify a time during the execution of the algorithm at which only few vertices were added greedily, yet, the majority of the vertices were contained in small separated components (and were, therefore, part of V_{Exact}), and only few vertices remain to be added greedily. Since the algorithm processes the vertices by increasing radius, this point in time can be translated to a threshold radius ρ in the hyperbolic disk (see Figure 1). Therefore, we divide the hyperbolic disk into two regions: an *inner disk* and an *outer band*, containing vertices with radii below and above ρ , respectively. The threshold ρ is chosen such that a hyperbolic random graph decomposes into small components after removing the inner disk. When adding the first vertex from the outer band, greedily, we can assume that the inner disk is empty (since vertices of smaller radii were chosen before or removed as part of a small component). At this point, the majority of the vertices in the outer band were contained in small components, which have been solved exactly. Therefore, we obtain a valid upper bound on $|V_{\text{Greedy}}|$, by counting the total number of vertices in the inner disk and adding the number of vertices in the outer band that are contained in components that are not solved exactly (i.e., components whose size exceeds $\tau \log \log(n)$). In the following, we show that both numbers are sublinear in n with high probability. Together with the fact that an optimal vertex cover on hyperbolic random graphs, asymptotically almost surely, contains $\Omega(n)$ vertices [10], this implies $|V_{\text{Greedy}}| \in o(|C_{\text{OPT}}|)$.

The main contribution of our analysis is the identification of small components in the outer band, which is done by discretizing it into sectors, such that an edge cannot extend beyond an empty sector (see Figure 1 (right)). The foundation of this analysis is the delicate

interplay between the angular width γ of these sectors and the threshold ρ that defines the outer band. Recall that ρ is used to represent the time in the execution of the algorithm at which the graph has been decomposed into small components. For our analysis we assume that all vertices seen before this point (all vertices in the inner disk) were added greedily. Therefore, if we choose ρ too large, we overestimate the actual number of greedily added vertices by too much. As a consequence, we want to choose ρ as small as possible. However, this conflicts our intentions for the choice of γ and its impact on ρ . Recall that the maximum angular distance between two vertices such that they are adjacent increases with decreasing radii (Equation (2)). Thus, in order to avoid edges that extend beyond an angular width of γ , we need to ensure that the radii of the vertices in the outer band are sufficiently large. That is, decreasing γ requires increasing ρ . However, we want to make γ as small as possible, in order to get a finer granularity in the discretization and, with that, a more accurate analysis of the component structure in the outer band. Therefore, γ and ρ need to be chosen such that the inner disk does not become too large, while ensuring that the discretization is granular enough to accurately detect components whose size depends on τ and n . To this end, we adjust the angular width of the sectors using a function $\gamma(n, \tau)$, which is defined as

$$\gamma(n, \tau) = \log(\tau \log^{(2)}(n) / (2 \log^{(3)}(n)^2)),$$

where $\log^{(i)}(n)$ denotes iteratively applying the log-function i times on n (e.g., $\log^{(2)}(n) = \log \log(n)$), and set

$$\rho = R - \log(\pi/2 \cdot e^{C/2} \gamma(n, \tau)),$$

where $R = 2 \log(n) + C$ is the radius of the hyperbolic disk.

In the following, we first show that the number of vertices in the inner disk is sublinear, with high probability, before analyzing the component structure in the outer band. This is mainly done by considering the random variables that denote the numbers of vertices in certain areas of the disk. We give proofs for their expected values throughout the paper. Tight concentration bounds can then be obtained using the previously mentioned Chernoff bound or, when the considered random variables are more involved, the method of (typical) bounded differences. These proofs can be found in the full version of the paper [7].

4.1 The Inner Disk

The inner disk \mathcal{I} contains all vertices whose radius is below the threshold ρ . The number of them that are added to the cover greedily is bounded by the number of all vertices in \mathcal{I} .

► **Lemma 5.** *Let G be a hyperbolic random graph on n vertices with power-law exponent $\beta = 2\alpha + 1$. With high probability, the number of vertices in \mathcal{I} is in $\mathcal{O}(n \cdot \gamma(n, \tau)^{-\alpha})$.*

Proof. We start by computing the expected number of vertices in \mathcal{I} and show concentration afterwards. To this end, we first compute the measure $\mu(\mathcal{I})$. The measure of a disk of radius r that is centered at the origin is given by $e^{-\alpha(R-r)}(1 + o(1))$ [17, Lemma 3.2]. Consequently, the expected number of vertices in \mathcal{I} is

$$\mathbb{E}[|V(\mathcal{I})|] = n\mu(\mathcal{I}) = \mathcal{O}(ne^{-\alpha(R-\rho)}) = \mathcal{O}(ne^{-\alpha \log(\pi/2 \cdot e^{C/2} \gamma(n, \tau))}) = \mathcal{O}(n \cdot \gamma(n, \tau)^{-\alpha}).$$

Since $\gamma(n, \tau) = \mathcal{O}(\log^{(3)}(n))$, this bound on $\mathbb{E}[|V(\mathcal{I})|]$ is $\omega(\log(n))$, and we can apply the Chernoff bound in Corollary 2 to conclude that $|V(\mathcal{I})| = \mathcal{O}(n \cdot \gamma(n, \tau)^{-\alpha})$ holds with probability $1 - \mathcal{O}(n^{-c})$ for any $c > 0$. ◀

Since $\gamma(n, \tau) = \omega(1)$, Lemma 5 shows that, with high probability, the number of vertices that are greedily added to the vertex cover in the inner disk is sublinear. Once the inner disk has been processed and removed, the graph has been decomposed into small components and the ones of size at most $\tau \log \log(n)$ have already been solved exactly. The remaining vertices that are now added greedily belong to large components in the outer band.

4.2 The Outer Band

To identify the vertices in the outer band that are contained in components whose size exceeds $\tau \log \log(n)$, we divide it into sectors of angular width $\gamma = \theta(\rho, \rho) = \pi \cdot \gamma(n, \tau) / n \cdot (1 + o(1))$, where $\theta(\rho, \rho)$ denotes the maximum angular distance between two vertices with radii ρ to be adjacent (see Equation (2)). This division is depicted in Figure 1 (right). The choice of γ (combined with the choice of ρ) has the effect that an edge between two vertices in the outer band cannot extend beyond an empty sector, i.e., a sector that does not contain any vertices, allowing us to use empty sectors as delimiters between components. To this end, we introduce the notion of *runs*, which are maximal sequences of non-empty sectors (grey in Figure 1 (right)). While a run can contain multiple components, the number of vertices in it denotes an upper bound on the combined sizes of the components that it contains.

To show that there are only few vertices in components whose size exceeds $\tau \log \log(n)$, we bound the number of vertices in runs that contain more than $\tau \log \log(n)$ vertices. For a given run this can happen for two reasons. First, it may contain many vertices if its angular interval is too large, i.e., it consists of too many sectors. This is unlikely, since the sectors are chosen sufficiently small, such that the probability for a given one to be empty is high. Second, while the angular width of the run is not too large, it contains too many vertices for its size. However, the vertices of the graph are distributed uniformly at random in the disk, making it unlikely that too many vertices are sampled into such a small area. To formalize this, we introduce a threshold w and distinguish between two types of runs: A *wide* run contains more than w sectors, while a *narrow* run contains at most w sectors. The threshold w is chosen such that the probabilities for a run to be wide and for a narrow run to contain more than $\tau \log \log(n)$ vertices are small. To this end, we set $w = e^{\gamma(n, \tau)} \cdot \log^{(3)}(n)$.

In the following, we first bound the number of vertices in wide runs. Afterwards, we consider narrow runs that contain more than $\tau \log \log(n)$ vertices. Together, this gives an upper bound on the number of vertices that are added greedily in the outer band.

4.2.1 Wide Runs

We refer to a sector that contributes to a wide run as a *widening sector*. In the following, we bound the number of vertices in all wide runs in three steps. First, we determine the expected number of all widening sectors. Second, based on the expected value, we show that the number of widening sectors is small, with high probability. Finally, we make use of the fact that the area of the disk covered by widening sectors is small, to show that the number of vertices sampled into the corresponding area is sublinear, with high probability.

Expected Number of Widening Sectors. Let n' denote the total number of sectors and let $S_1, \dots, S_{n'}$ be the corresponding sequence. For each sector S_k , we define the random variable S_k indicating whether S_k contains any vertices, i.e., $S_k = 0$ if S_k is empty and $S_k = 1$ otherwise. The sectors in the disk are then represented by a circular sequence of indicator random variables $S_1, \dots, S_{n'}$, and we are interested in the random variable W that denotes the sum of all runs of 1s that are longer than w . In order to compute $\mathbb{E}[W]$, we first compute the total number of sectors, as well as the probability for a sector to be empty or non-empty.

► **Lemma 6.** *Let G be a hyperbolic random graph on n vertices. Then, the number of sectors of width $\gamma = \theta(\rho, \rho)$ is $n' = 2n/\gamma(n, \tau) \cdot (1 \pm o(1))$.*

Proof. Since all sectors have equal angular width $\gamma = \theta(\rho, \rho)$, we can use Equation (2) to compute the total number of sectors as $n' = 2\pi/\theta(\rho, \rho) = \pi e^{-R/2+\rho}(1 \pm \Theta(e^{R-2\rho}))^{-1}$. By substituting $\rho = R - \log(\pi/2 \cdot e^{C/2}\gamma(n, \tau))$ and $R = 2\log(n) + C$, we obtain

$$n' = \frac{\pi e^{R/2}}{\pi/2 \cdot e^{C/2}\gamma(n, \tau)} (1 \pm \Theta(e^{-R}\gamma(n, \tau)^2))^{-1} = 2n/\gamma(n, \tau) \cdot (1 \pm \Theta((\gamma(n, \tau)/n)^2))^{-1}.$$

It remains to simplify the error term. Note that $\gamma(n, \tau) = \mathcal{O}(\log^{(3)}(n))$. Consequently, the error term is equivalent to $(1 \pm o(1))^{-1}$. Finally, it holds that $1/(1+x) = 1 - \Theta(x)$ is valid for $x = \pm o(1)$, according to Lemma 3. ◀

► **Lemma 7.** *Let G be a hyperbolic random graph on n vertices and let \mathcal{S} be a sector of angular width $\gamma = \theta(\rho, \rho)$. For sufficiently large n , the probability that \mathcal{S} contains at least one vertex is bounded by*

$$1 - e^{-\gamma(n, \tau)/4} \leq \Pr[V(\mathcal{S}) \neq \emptyset] \leq e^{-\left(e^{-\gamma(n, \tau)}\right)}.$$

We are now ready to bound the expected number of widening sectors, i.e., sectors that are part of wide runs. To this end, we aim to apply the following lemma.

► **Lemma 8** ([23, Proposition 4.3¹]). *Let $S_1, \dots, S_{n'}$ denote a circular sequence of independent indicator random variables, such that $\Pr[S_k = 1] = p$ and $\Pr[S_k = 0] = 1 - p = q$, for all $k \in \{1, \dots, n'\}$. Furthermore, let W denote the sum of the lengths of all success runs of length at least $w \leq n'$. Then, $\mathbb{E}[W] = n'p^w(wq + p)$.*

We note that the indicator random variables $S_1, \dots, S_{n'}$ are *not* independent on hyperbolic random graphs. To overcome this issue, we compute the expected value of W on hyperbolic random graphs with n vertices in expectation (see Section 2) and subsequently derive a probabilistic bound on W for hyperbolic random graphs.

► **Lemma 9.** *Let G be a hyperbolic random graph with n vertices in expectation and let W denote the number of widening sectors. Then,*

$$\mathbb{E}[W] \leq \frac{2^{1/4} \cdot \tau^{3/4} \cdot n}{\gamma(n, \tau) \cdot \log^{(2)}(n)^{1/4} \cdot \log^{(3)}(n)^{1/2}} (1 \pm o(1)).$$

Proof. A widening sector is part of a run of more than $w = e^{\gamma(n, \tau)} \cdot \log^{(3)}(n)$ consecutive non-empty sectors. To compute the expected number of widening sectors, we apply Lemma 8. To this end, we use Lemma 6 to bound the total number of sectors n' and bound the probability $p = \Pr[S_k = 1]$ (i.e., the probability that sector \mathcal{S}_k is not empty) as $p \leq \exp(-(e^{-\gamma(n, \tau)}))$, as

¹ The original statement has been adapted to fit our notation. We use n', w , and W to denote the total number of random variables, the threshold for long runs, and the sum of their lengths, respectively. They were previously denoted by n, k , and S , respectively. In the original statement $s = 0$ indicates that the variables are distributed independently and identically, and c indicates that the sequence is circular.

well as the complementary probability $q = 1 - p \leq e^{-\gamma(n,\tau)/4}$ using Lemma 7. We obtain

$$\begin{aligned}\mathbb{E}[W] &= n' p^{(w+1)} ((w+1)q + p) \\ &\leq \frac{2n}{\gamma(n,\tau)} (1 \pm o(1)) \cdot e^{-((w+1)e^{-\gamma(n,\tau)})} \cdot \left((w+1)e^{-\frac{\gamma(n,\tau)}{4}} + 1 \right) \\ &\leq \frac{2n}{\gamma(n,\tau)} e^{(-e^{\gamma(n,\tau)} \log^{(3)}(n) e^{-\gamma(n,\tau)})} \left((e^{\gamma(n,\tau)} \log^{(3)}(n) + 1) e^{-\frac{\gamma(n,\tau)}{4}} + 1 \right) (1 \pm o(1)) \\ &= \frac{2ne^{3/4 \cdot \gamma(n,\tau)} \log^{(3)}(n)}{\gamma(n,\tau) \cdot \log^{(2)}(n)} \left(1 + \frac{1}{e^{\gamma(n,\tau)} \log^{(3)}(n)} + \frac{1}{e^{3/4 \cdot \gamma(n,\tau)} \log^{(3)}(n)} \right) (1 \pm o(1)).\end{aligned}$$

Since $\gamma(n,\tau) = \omega(1)$, the first error term can be simplified as $(1 + o(1))$. Additionally, we can substitute $\gamma(n,\tau) = \log(\tau \log^{(2)}(n)/(2 \log^{(3)}(n)^2))$ to obtain

$$\mathbb{E}[W] \leq 2^{1/4} \frac{\tau^{3/4} \cdot n \cdot \log^{(3)}(n)}{\gamma(n,\tau) \cdot \log^{(2)}(n)} \cdot \frac{\log^{(2)}(n)^{3/4}}{\log^{(3)}(n)^{3/2}} \cdot (1 \pm o(1)).$$

Further simplification then yields the claim. \blacktriangleleft

Concentration Bound on the Number of Widening Sectors. Lemma 9 bounds the expected number of widening sectors and it remains to show that this bound holds with high probability. To this end, we first determine under which conditions the sum of long success runs in a circular sequence of indicator random variables can be bounded with high probability in general. Afterwards, we show that these conditions are met for our application.

► **Lemma 10.** *Let $S_1, \dots, S_{n'}$ denote a circular sequence of independent indicator random variables and let W denote the sum of the lengths of all success runs of length at least $1 \leq w \leq n'$. If $g(n') = \omega(w\sqrt{n' \log(n')})$ is an upper bound on $\mathbb{E}[W]$, then $W = \mathcal{O}(g(n'))$ holds with probability $1 - \mathcal{O}((n')^{-c})$ for any constant c .*

► **Lemma 11.** *Let G be a hyperbolic random graph on n vertices. Then, with probability $1 - \mathcal{O}(n^{-c})$ for any constant $c > 0$, the number of widening sectors W is bounded by*

$$W = \mathcal{O} \left(\frac{\tau^{3/4} \cdot n}{\gamma(n,\tau) \cdot \log^{(2)}(n)^{1/4} \cdot \log^{(3)}(n)^{1/2}} \right).$$

Number of Vertices in Wide Runs. Let \mathcal{W} denote the area of the disk covered by all widening sectors. By Lemma 11 the total number of widening sectors is small, with high probability. As a consequence, \mathcal{W} is small as well and we can derive that the size of the vertex set $V(\mathcal{W})$ containing all vertices in all widening sectors is sublinear with high probability.

► **Lemma 12.** *Let G be a hyperbolic random graph on n vertices. Then, with high probability, the number of vertices in wide runs is bounded by*

$$|V(\mathcal{W})| = \mathcal{O} \left(\frac{\tau^{3/4} \cdot n}{\log^{(2)}(n)^{1/4} \cdot \log^{(3)}(n)^{1/2}} \right).$$

It remains to bound the number of vertices in large components contained in narrow runs.

4.2.2 Narrow Runs

In the following, we differentiate between *small* and *large* narrow runs, containing at most and more than $\tau \log \log(n)$ vertices, respectively. To obtain an upper bound on the number N of vertices in all large narrow runs, we determine the area \mathcal{N} of the disk that is covered by them. We start by computing the expected number of vertices contained in a single narrow run from which we can derive that the probability for a narrow run to be large is low.

Expected Number of Vertices in Large Narrow Runs.

► **Lemma 13.** *Let G be a hyperbolic random graph on n vertices and let \mathcal{R} be a narrow run. Then, $\mathbb{E}[|V(\mathcal{R})|] \leq 1/2 \cdot e^{\gamma(n,\tau)} \log^{(3)}(n) \gamma(n,\tau) (1 \pm o(1))$.*

Proof. A narrow run consists of at most $w = e^{\gamma(n,\tau)} \log^{(3)}(n)$ sectors. Since the angular coordinates of the vertices are distributed uniformly at random and since we partitioned the disk into n' disjoint sectors of equal width, we can derive an upper bound on the expected number of vertices in \mathcal{R} as $\mathbb{E}[|V(\mathcal{R})|] \leq nw/n'$. As $n' = 2n/\gamma(n,\tau)(1 \pm o(1))$ according to Lemma 6, we have

$$\mathbb{E}[|V(\mathcal{R})|] \leq 1/2 \cdot e^{\gamma(n,\tau)} \log^{(3)}(n) \gamma(n,\tau) (1 \pm o(1))^{-1}.$$

Since $1/(1+x) = 1 - \Theta(x)$ for $x = \pm o(1)$ (Lemma 3), we obtain the claimed bound. ◀

Using this upper bound, we can bound the probability that the number of vertices in a narrow run exceeds the threshold $\tau \log \log(n)$ by a certain amount.

► **Lemma 14.** *Let G be a hyperbolic random graph on n vertices and let \mathcal{R} be a narrow run. For $k > \tau \log \log(n)$ and n large enough, it holds that $\Pr[|V(\mathcal{R})| = k] \leq e^{-k/18}$.*

We are now ready to compute the expected number of vertices in all large narrow runs.

► **Lemma 15.** *Let G be a hyperbolic random graph. Then, the expected number of vertices in all large narrow runs is bounded by*

$$\mathbb{E}[N] = \mathcal{O} \left(\frac{\tau \cdot n \cdot \log^{(2)}(n)}{\gamma(n,\tau) \log(n)^{\tau/18}} \right).$$

Proof. Let n'' denote the total number of narrow runs. We can compute the number of vertices in all large narrow runs, by summing over all narrow runs $\mathcal{R}_1, \dots, \mathcal{R}_{n''}$ and discarding the ones that are not large. That is,

$$N = \sum_{i=1}^{n''} |V(\mathcal{R}_i)| \cdot \mathbb{1}_{|V(\mathcal{R}_i)| > \tau \log^{(2)}(n)}.$$

Consequently, the expected value of N is given by

$$\mathbb{E}[N] = \sum_{i=1}^{n''} \mathbb{E} \left[|V(\mathcal{R}_i)| \cdot \mathbb{1}_{|V(\mathcal{R}_i)| > \tau \log^{(2)}(n)} \right] = \sum_{i=1}^{n''} \sum_{k=\tau \log^{(2)}(n)+1}^n k \cdot \Pr[|V(\mathcal{R}_i)| = k].$$

Lemma 14 gives a valid upper bound on $\Pr[|V(\mathcal{R}_i)| = k]$ for all $i \in \{1, \dots, n''\}$. Furthermore, the number of narrow runs n'' is bounded by the number of sectors n' . Therefore, we obtain

$$\mathbb{E}[N] \leq n' \sum_{k=\tau \log^{(2)}(n)+1}^n k \cdot e^{-k/18}.$$

To get an upper bound, we replace the sum with an integral, which yields

$$\begin{aligned}\mathbb{E}[N] &\leq n' \int_{\tau \log^{(2)}(n)}^n k e^{-\frac{k}{18}} dk \\ &\leq n' \left[18e^{-\tau/18 \log^{(2)}(n)} (\tau \log^{(2)}(n) + 18) - 18e^{-n/18} (n + 18) \right] \\ &\leq 18n' \cdot \frac{\tau \log^{(2)}(n) + 18}{\log(n)^{\tau/18}},\end{aligned}$$

where the last inequality holds since $e^{-n/18}(n+18) \geq 0$. Substituting $n' = 2n/\gamma(n, \tau)(1 \pm o(1))$ (Lemma 6) and more simplification yield the claim. \blacktriangleleft

Concentration Bound on the Number of Vertices in Large Narrow Runs. To show that the actual number of vertices in large narrow runs is not much larger than the expected value, we apply the method of typical bounded differences [29]. (See the full version of the paper for a detailed explanation [7].) To this end, we consider N as a function of the positions of the vertices and bound the effect that changing a single position can have on this function. It is easy to see, that this effect is the largest, when the change splits a wide run \mathcal{R} into two large narrow runs. If \mathcal{R} contained a lot of vertices, the impact on N is large. However, since the vertices are distributed uniformly, it is very unlikely that a run that can be split into two narrow runs contains many vertices.

► **Lemma 16.** *Let G be a hyperbolic random graph. Then, each run of length at most $2w + 1$ contains at most $\mathcal{O}(\log(n))$ vertices with probability $1 - \mathcal{O}(n^{-c})$ for any constant c .*

The method of typical bounded differences now allows us to focus on this case and to milden the impact of the worst case changes as they occur with small probability. Consequently, we can show that the number of vertices in large narrow runs is sublinear with high probability.

► **Lemma 17.** *Let G be a hyperbolic random graph. Then, with high probability, the number of vertices in large narrow runs is bounded by*

$$N = \mathcal{O}\left(\frac{\tau \cdot n \cdot \log^{(2)}(n)}{\gamma(n, \tau) \log(n)^{\tau/18}}\right).$$

4.3 The Complete Disk

In the previous subsections we determined the number of vertices that are greedily added to the vertex cover in the inner disk and outer band, respectively. Before proving our main theorem, we are now ready to prove a slightly stronger version that shows how the parameter τ can be used to obtain a tradeoff between approximation performance and running time.

► **Theorem 18.** *Let G be a hyperbolic random graph on n vertices with power-law exponent $\beta = 2\alpha + 1$ and let $\tau > 0$ be constant. Given the radii of the vertices, an approximate vertex cover of G can be computed in time $\mathcal{O}(m \log(n)^\tau + n \log(n))$, such that the approximation factor is $(1 + \mathcal{O}(\gamma(n, \tau)^{-\alpha}))$ asymptotically almost surely.*

Proof. *Running Time.* We start by sorting the vertices of the graph in order of increasing radius, which can be done in time $\mathcal{O}(n \log(n))$. Afterwards, we iterate them and perform the following steps for each encountered vertex v . We add v to the cover, remove it from the graph, and identify connected components of size at most $\tau \log \log(n)$ that were separated by the removal. The first two steps can be performed in time $\mathcal{O}(1)$ and $\mathcal{O}(\deg(v))$, respectively.

Identifying and solving small components is more involved. Removing v can split the graph into at most $\deg(v)$ components, each containing a neighbor u of v . Such a component can be identified by performing a breadth-first search (BFS) starting at u . Each BFS can be stopped as soon as it encounters more than $\tau \log \log(n)$ vertices. The corresponding subgraph contains at most $(\tau \log \log(n))^2$ edges. Therefore, a single BFS takes time $\mathcal{O}(\log \log(n)^2)$. Whenever a component of size at most $n_c = \tau \log \log(n)$ is found, we compute a minimum vertex cover for it in time $1.1996^{n_c} \cdot n_c^{\mathcal{O}(1)}$ [30]. Since $n_c^{\mathcal{O}(1)} = \mathcal{O}((e/1.1996)^{n_c})$, this running time is bounded by $\mathcal{O}(e^{n_c}) = \mathcal{O}(\log(n)^\tau)$. Consequently, the time required to process each neighbor of v is $\mathcal{O}(\log(n)^\tau)$. Since this is potentially performed for all neighbors of v , the running time of this third step can be bounded by introducing an additional factor of $\deg(v)$. We then obtain the total running time $T(n, m, \tau)$ of the algorithm by summing the running times of all three steps over all vertices, which yields

$$\begin{aligned} T(n, m, \tau) &= \sum_{v \in V} \mathcal{O}(1) + \mathcal{O}(\deg(v)) + \deg(v) \cdot \mathcal{O}(\log(n)^\tau) \\ &= \mathcal{O} \left(\log(n)^\tau \cdot \sum_{v \in V} \deg(v) \right) = \mathcal{O}(m \log(n)^\tau). \end{aligned}$$

Approximation Ratio. As argued before, we obtain a valid vertex cover for the whole graph, if we take all vertices in V_{Greedy} together with a vertex cover C_{Exact} of $G[V_{\text{Exact}}]$. The approximation ratio of the resulting cover is then given by the quotient

$$\delta = \frac{|V_{\text{Greedy}}| + |C_{\text{Exact}}|}{|C_{\text{OPT}}|},$$

where C_{OPT} denotes an optimal solution. Since all components in $G[V_{\text{Exact}}]$ are solved optimally and since any minimum vertex cover for the whole graph induces a vertex cover on $G[V']$ for any vertex subset $V' \subseteq V$, it holds that $|C_{\text{Exact}}| \leq |C_{\text{OPT}}|$. Therefore, the approximation ratio can be bounded by $\delta \leq 1 + |V_{\text{Greedy}}|/|C_{\text{OPT}}|$. To bound the number of vertices in V_{Greedy} , we add the number of vertices in the inner disk \mathcal{I} , as well as the numbers of vertices in the outer band that are contained in the area \mathcal{W} that is covered by wide runs and the area \mathcal{N} that is covered by large narrow runs. That is,

$$\delta \leq 1 + \frac{|V(\mathcal{I})| + |V(\mathcal{W})| + |V(\mathcal{N})|}{|C_{\text{OPT}}|}.$$

Upper bounds on $|V(\mathcal{I})|$, $|V(\mathcal{W})|$, and $|V(\mathcal{N})|$ that hold with high probability are given by Lemmas 5, 12, and 17, respectively. Furthermore, it was previously shown that the size of a minimum vertex cover on a hyperbolic random graph is $|C_{\text{OPT}}| = \Omega(n)$, asymptotically almost surely [10, Theorems 4.10 and 5.8]. We obtain

$$\delta = 1 + \mathcal{O} \left(\frac{1}{\gamma(n, \tau)^\alpha} + \frac{\tau^{3/4}}{\log^{(2)}(n)^{1/4} \cdot \log^{(3)}(n)^{1/2}} + \frac{\tau \cdot \log^{(2)}(n)}{\gamma(n, \tau) \log(n)^{\tau/18}} \right).$$

Since $\gamma(n, \tau) = \mathcal{O}(\log^{(3)}(n))$, the first summand dominates asymptotically. \blacktriangleleft

► Theorem 4. *Let G be a hyperbolic random graph on n vertices. Given the radii of the vertices, an approximate vertex cover of G can be computed in time $\mathcal{O}(m \log(n))$, such that the approximation ratio is $(1 + o(1))$ asymptotically almost surely.*

Proof. By Theorem 18 we can compute an approximate vertex cover in time $\mathcal{O}(n \log(n) + m \log(n)^\tau)$, such that the approximation factor is $1 + \mathcal{O}(\gamma(n, \tau)^{-\alpha})$, asymptotically almost surely. We obtain the claimed bound on the running time by choosing $\tau = 1$. Furthermore, since $\gamma(n, 1) = \omega(1)$ and $\alpha \in (1/2, 1)$, the resulting approximation factor is $(1 + o(1))$. \blacktriangleleft

References

- 1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics*, pages 62–69, 2004.
- 2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.
- 3 Eric Angel, Romain Campigotto, and Christian Laforest. Implementation and comparison of heuristics for the vertex cover problem on huge graphs. In *Experimental Algorithms*, pages 39–50, 2012.
- 4 Yonatan Bilu and Nathan Linial. Are stable instances easy? *Comb. Probab. Comput.*, 21(5):643–660, 2012. doi:10.1017/S0963548312000193.
- 5 Thomas Bläsius, Philipp Fischbeck, Tobias Friedrich, and Maximilian Katzmann. Solving Vertex Cover in Polynomial Time on Hyperbolic Random Graphs. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, pages 25:1–25:14, 2020. doi:10.4230/LIPIcs.STACS.2020.25.
- 6 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 *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 20:1–20:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.20.
- 7 Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann. Efficiently approximating vertex cover on scale-free networks with underlying hyperbolic geometry. *CoRR*, abs/2010.02787, 2020. URL: <https://arxiv.org/abs/2010.02787>.
- 8 Marián Boguná, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1:62, 2010. doi:10.1038/ncomms1063.
- 9 Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrak. Stability and Recovery for Independence Systems. In *25th Annual European Symposium on Algorithms (ESA 2017)*, pages 26:1–26:15, 2017. doi:10.4230/LIPIcs.ESA.2017.26.
- 10 Ankit Chauhan, Tobias Friedrich, and Ralf Rothenberger. Greed is Good for Deterministic Scale-Free Networks. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, pages 33:1–33:15, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.33.
- 11 Mariana O. Da Silva, Gustavo A. Gimenez-Lugo, and Murilo V. G. Da Silva. Vertex cover in complex networks. *Int. J. Mod. Phys. C*, 24(11):1350078, 2013. doi:10.1142/S0129183113500782.
- 12 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2012.
- 13 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. Vertex cover meets scheduling. *Algorithmica*, 74:1148–1173, 2016. doi:10.1007/s00453-015-9992-y.
- 14 Eric Filiol, Edouard Franc, Alessandro Gubbioli, Benoit Moquet, and Guillaume Roblot. Combinatorial optimisation of worm propagation on an unknown network. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 1:2931–2937, 2007.
- 15 Guillermo García-Pérez, Marián Boguná, Antoine Allard, and M. Serrano. The hidden hyperbolic geometry of international trade: World trade atlas 1870–2013. *Scientific Reports*, 6:33441, 2016. doi:10.1038/srep33441.
- 16 Mikael Gast and Mathias Hauptmann. Approximability of the vertex cover problem in power-law graphs. *Theoretical Computer Science*, 516:60–70, 2014. doi:10.1016/j.tcs.2013.11.004.

- 17 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *Automata, Languages, and Programming*, pages 573–585. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-31585-5_51.
- 18 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.
- 19 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
- 20 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2 - \varepsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.
- 21 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.
- 22 Anton Krohmer. *Structures & algorithms in Hyperbolic Random Graphs*. doctoralthesis, University of Potsdam, 2016.
- 23 Frosso S. Makri, Andreas N. Philippou, and Zaharias M. Psillakis. Success run statistics defined on an urn model. *Advances in Applied Probability*, 39(4):991–1019, 2007. doi:10.1239/aap/1198177236.
- 24 Tobias Müller and Merlijn Staps. The diameter of KPKVB random graphs. *Advances in Applied Probability*, 51:358–377, 2019. doi:10.1017/apr.2019.23.
- 25 Kihong Park and Walter Williger. *The Internet As a Large-Scale Complex System*. Oxford University Press, Inc., 2005.
- 26 K. Subhash, D. Minzer, and M. Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 592–601, 2018.
- 27 Kevin Verbeek and Subhash Suri. Metric embedding, hyperbolic space, and social networks. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG’14, page 501–510, 2014. doi:10.1145/2582112.2582139.
- 28 André L. Vignatti and Murilo V.G. da Silva. Minimum vertex cover in generalized random graphs with power law degree distribution. *Theoretical Computer Science*, 647:101–111, 2016. doi:10.1016/j.tcs.2016.08.002.
- 29 Lutz Warnke. On the method of typical bounded differences. *Combinatorics, Probability and Computing*, 25(2):269–299, 2016. doi:10.1017/S0963548315000103.
- 30 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

Efficiently Computing Maximum Flows in Scale-Free Networks

Thomas Bläsius ✉

Karlsruhe Institute of Technology, Germany

Tobias Friedrich ✉

Hasso Plattner Institute, Universität Potsdam, Germany

Christopher Weyand ✉

Karlsruhe Institute of Technology, Germany

Abstract

We study the maximum-flow/minimum-cut problem on scale-free networks, i.e., graphs whose degree distribution follows a power-law. We propose a simple algorithm that capitalizes on the fact that often only a small fraction of such a network is relevant for the flow. At its core, our algorithm augments Dinitz’s algorithm with a balanced bidirectional search. Our experiments on a scale-free random network model indicate sublinear run time. On scale-free real-world networks, we outperform the commonly used highest-label Push-Relabel implementation by up to two orders of magnitude. Compared to Dinitz’s original algorithm, our modifications reduce the search space, e.g., by a factor of 275 on an autonomous systems graph.

Beyond these good run times, our algorithm has an additional advantage compared to Push-Relabel. The latter computes a preflow, which makes the extraction of a minimum cut potentially more difficult. This is relevant, for example, for the computation of Gomory-Hu trees. On a social network with 70 000 nodes, our algorithm computes the Gomory-Hu tree in 3 seconds compared to 12 minutes when using Push-Relabel.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases graphs, flow, network, scale-free

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.21

Related Version *Full Version*: <https://arxiv.org/abs/2009.09678> [5]

Supplementary Material *Software (code and data)*: <https://github.com/chistopher/scale-free-flow>; archived at [swh:1:dir:48f8148b8dbabab86dfd6e86ab7dd5dffaeb3cd0](https://www.swh.io/dir/48f8148b8dbabab86dfd6e86ab7dd5dffaeb3cd0)

1 Introduction

The maximum flow problem is arguably one of the most fundamental graph problems that regularly appears as a subtask in various applications [2, 29, 32]. The go-to general-purpose algorithm for computing flows in practice is the highest-label Push-Relabel algorithm by Cherkassky and Goldberg [11], which is also part of the boost graph library [30]. Beyond that, the BK-algorithm by Boykov and Kolmogorov [8] or its later iteration [17] should be used for instances appearing in computer vision. Our main goal in this paper is to provide a flow algorithm tailored towards *scale-free* networks. Such networks are characterized by their heavy-tailed degree distribution resembling a power-law, i.e., they are sparse with few vertices of comparatively high degree and many vertices of low degree.

At its core, our algorithm is a variant of Dinitz’s algorithm [13], which is an augmenting path algorithm that iteratively increases the flow along collections of shortest paths in the residual network. In each iteration, at least one edge on every shortest path gets saturated, thereby increasing the distance between source and sink in the residual network. To exploit the structure of scale-free networks, we make use of the facts that, firstly, shortest paths tend



© Thomas Bläsius, Tobias Friedrich, and Christopher Weyand;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



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

to span only a small fraction of such networks, and secondly, a balanced bidirectional breadth first search is able to find the shortest paths very efficiently [7, 6]. Using a bidirectional search to compute shortest paths in Dinitz’s algorithm directly translates this efficiency to the first iteration, as the residual network initially coincides with the flow network. Though the structure of the residual network changes in later iterations, our experiments show that the run time improvements achieved by using a bidirectional search remain high.

Scaling experiments with geometric inhomogeneous random graphs (GIRGs)¹ [9] indicate that the flow computation of our algorithm runs in sublinear time. In comparison, previous algorithms (Push-Relabel, BK, and unidirectional Dinitz) require slightly super-linear time. This is also reflected in the high speedups we achieve on real-world scale-free networks.

With the flow computation itself being so efficient, the total run time for computing the maximum flow for a single source-sink pair in a scale-free network is heavily dominated by loading the graph and building data structures. Thus, our algorithm is particularly relevant when we have to compute multiple flows in the same network. This is, e.g., the case when computing the Gomory-Hu tree [20] of a network. The Gomory-Hu tree is a compact representation of the minimum s - t cuts for all source-sink pairs (s, t) . It can be computed with Gusfield’s algorithm [21] using $n - 1$ flow computations in a network with n vertices. Using our bidirectional flow algorithm as the subroutine for flow computations in Gusfield’s algorithm lets us compute the Gomory-Hu tree of, e.g., the `soc-slashdot` instance with 70k nodes and 360k edges in only 2.6s. In this context, we observe that the Push-Relabel algorithm is also very efficient in computing the flow values by computing a preflow. However, converting this to a flow or extracting a cut from it takes significantly more time.

Contribution. Our findings can be summarized in the following main contributions.

- We provide a simple and efficient flow-algorithm that significantly outperforms previous algorithms on scale-free networks.
- It’s efficiency on non-scale-free instances makes it a potential replacement for the Push-Relabel algorithm for general-purpose flow computations.
- Our algorithm is well suited to compute the Gomory-Hu tree of large instances.
- In contrast to previous observations [11, 12], situations exist where computing a flow with the Push-Relabel algorithm is significantly more expensive than computing a preflow.

Related Work. We briefly discuss only the work most related to our result. For a more extensive overview on the topic of flows, we refer to the survey by Goldberg and Tarjan [19].

Our algorithm is based on Dinitz’s Algorithm [13], which belongs to the family of *augmenting path algorithms* originating from the Ford-Fulkerson algorithm [16]. Augmenting path algorithms use the *residual network* to represent the remaining capacities and iteratively increase the flow by augmenting it with paths from source to sink in the residual network, until no such path exists. At every point in time, a valid flow is known and at the end of execution, non-reachability in the residual network certifies maximality.

From this perspective, the *Push-Relabel algorithm* [18] does the reverse. At every point in time, the sink is not reachable from the source in the residual network, thereby guaranteeing maximality, while the object maintained throughout the algorithm is a so-called *preflow* and the algorithm stops once the preflow is actually a flow. This is achieved using two operations

¹ GIRGs are a generative network model closely related to hyperbolic random graphs [25]. They resemble real-world networks in regards to important properties such as degree distribution, clustering, and distances.

push and *relabel*; hence the name. Different variants of the Push-Relabel algorithm mainly differ with regards to the order in which operations are applied. A strategy performing well in practice is the highest-label strategy [11]. The extensive empirical study by Ahuja et al. [1] on ten different algorithms shows that the highest-label Push-Relabel algorithm indeed performs the best out of the ten. The only small caveat with these experiments is the fact that they are based on artificial networks that are specifically generated to pose difficult instances. Our experiments show that the structure of the instance matters in the sense that it impacts different algorithms differently; potentially yielding different rankings on different types of instances. The so-called pseudoflow algorithm by Hochbaum [23] was later shown to slightly outperform (low single-digit speedups on most instances) the highest-label Push-Relabel algorithm; again based on artificial instances [10].

Boykov and Kolmogorov [8] gave an algorithm tailored specifically towards instances that appear in computer vision; outperforming Push-Relabel on these instances. It was later refined by Goldberg et al. [17]. Most related to our studies is the work by Halim et al. [22] who developed a distributed flow algorithm for MapReduce on huge social networks.

2 Network Flows and Dinitz's Algorithm

Network Flows. A flow network is a directed graph $G = (V, E)$ with source and sink vertices $s, t \in V$, and a capacity function $c : V \times V \rightarrow \mathbb{N}$ with $c(u, v) = 0$ if $(u, v) \notin E$. A *flow* f on G is a function $f : V \times V \rightarrow \mathbb{Z}$ satisfying three constraints: (I) capacity $f(u, v) \leq c(u, v)$ (II) asymmetry $f(u, v) = -f(v, u)$ and (III) conservation $\sum_{v \in V} f(u, v) = 0$ for $u \in V \setminus \{s, t\}$. We call an edge $(u, v) \in E$ *saturated* if $f(u, v) = c(u, v)$. Denote the *value* of a flow f as $\sum_{v \in V} f(s, v)$. The maximum flow problem, *max-flow* for short, is the problem of finding a flow of maximum value.

Given a flow f in G , we define a network G_f called the *residual network*. G_f has the same set of nodes and contains the directed edge (u, v) if $f(u, v) < c(u, v)$. The capacity c' of edges in G_f is given by the residual capacity in the original network, i.e., $c'(u, v) = c(u, v) - f(u, v)$. An s - t path in G_f is called an *augmenting path*.

Dinitz's Algorithm. Let $d_s(v)$ be the distance from s to vertex v in G_f . We define a subgraph of G_f called the *layered network* by restricting the edge set to edges (u, v) of G_f for which $d_s(u) + 1 = d_s(v)$, i.e., edges that increase the distance to the source. We call a flow *blocking* if every s - t path contains at least one edge that is saturated by this flow, i.e., there is no augmenting path.

Dinitz's algorithm (see Algorithm 1) groups augmentations into rounds. It augments a set of edges that constitutes a blocking flow of the layered network in each round. One can find such a set of edges by iteratively augmenting s - t paths in the layered network until source and sink become disconnected. After augmenting a blocking flow, the distance between the terminals in the residual network strictly increases.

Algorithm 1 Dinitz's Algorithm.

```

1 while  $s$ - $t$  path in residual network do
2   build layered network
3   while  $s$ - $t$  path in layered network do
4     augment flow with  $s$ - $t$  path

```

Asymptotic Running Time. To better understand how our modifications impact the run time, we briefly sketch how Dinitz running time of $O(n^2m)$ is obtained. Since $d_s(t)$ increases each round, the number of rounds is bounded by $n - 1$. Each round consists of two stages: building the layered network and augmenting a blocking flow. The layered network can be constructed in $O(m)$ using a breadth-first search (BFS). Finding the blocking flow is done with a repeated graph traversal, usually using a depth-first search (DFS). The number of found paths is bounded by m , because each found path saturates at least one edge, removing it from the layered network. A single DFS can be done in amortized $O(n)$ time as follows. Edges that are not part of an s - t path in the layered network do not need to be looked at more than once during one round. This is achieved by remembering for each node which edges of the layered network were already found to have no remaining path to the sink. Each subsequent DFS will start where the last one left off. Thus, per round, the depth-first searches have a combined search space of $O(m)$, while each individual search additionally visits the nodes on one s - t path which is $O(n)$.

Practical Performance. In our experiments $d_s(t)$ remains mostly below 10, implying that the number of rounds is significantly lower than $n - 1$. Also, the number of found augmenting paths during one rounds is far below m . In unweighted networks, for example, a DFS saturates all edges of the found path resulting in a bound of $O(m)$ to find a blocking flow. Dinitz's algorithm has a tight upper bound of $O(n^{2/3}m)$ in unweighted networks [14, 24].

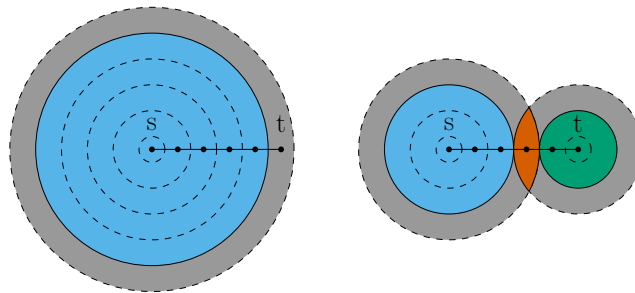
3 Improving Dinitz on Scale-Free Networks

We adapt a common Dinitz implementation² to exploit the specific structure of scale-free networks. We achieve a significant speedup by using the fact that a flow and cut respectively often depend only on a small fraction of the network. The following three modifications each tackle a performance bottleneck.

Bidirectional Search. Recently, sublinear running time was shown for balanced bidirectional search in a scale-free network model [6, 7]. We use a bidirectional breadth-first-search to compute the distances that define the layered network during each round of Dinitz's algorithm. A forward search is performed from the source and a backward search from the sink, each time advancing the search that incurs the lower cost to advance one layer. A shortest s - t path is found when a vertex is discovered that was already seen from the other direction. Note that, for our purpose, the bidirectional search has to finish the current layer when such a vertex is discovered, because all shortest paths must be found. Figure 1 visualizes the difference in explored vertices between a normal and a bidirectional BFS. The augmentations with DFS are restricted to the visited part of the layered network, meaning the search space of the BFS plus the next layer.

The distance labeling obtained by the bidirectional BFS requires a change to the DFS. The purpose of the layered network is to contain all edges on shortest s - t paths. The DFS identifies edges (u, v) of the layered network by checking if they increase the distance from the source, i.e., $d_s(u) + 1 = d_s(v)$. However, we no longer obtain the distances from the source for all relevant vertices. For vertices processed by the backward search, distances to the sink $d_t(v)$ are known instead. To resolve the problem, we allow edges that either increase distance from the source or decrease distance to the sink, i.e., $d_s(u) + 1 = d_s(v)$

² <https://cp-algorithms.com/graph/dinic.html>



■ **Figure 1** Search space of a breadth-first search from a source s to a sink t unidirectional (left) and bidirectional (right). The blue area represents the vertices that are explored, i.e., whose outgoing edges were scanned, by the forward search and the green area the backward search. In the gray area are vertices that are seen during exploration of the last layer, but not yet explored. Vertices in the intersection of the upcoming layers of the backward and forward search are marked orange.

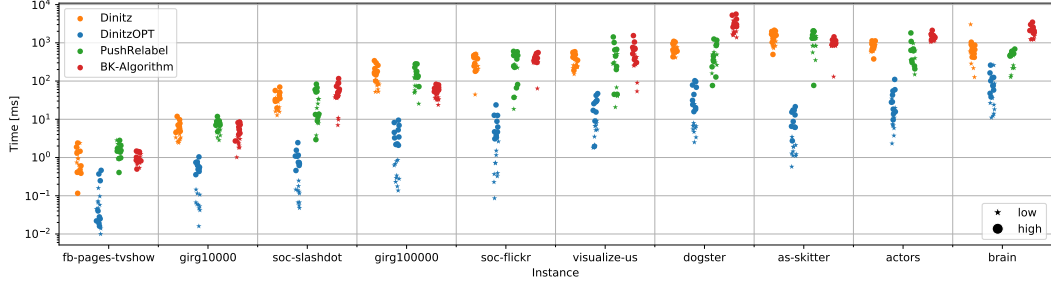
or $d_t(u) - 1 = d_t(v)$. This deviates from the definition of the layered network. But since edges on shortest s - t paths must both, increase the distance from the source and decrease the distance to the sink, we do not miss any relevant edges.

Time Stamps. The bidirectional search reduces the search space of the breadth-first search and depth-first search substantially, potentially to sublinear. The initialization, however, still requires linear time. It includes distances from the source and to the sink and one progress counter per node for the augmentations. To avoid the linear initializations, we introduce time stamps to indicate if a vertex was seen during the current round. The initialization of distances and counters is done lazily as vertices are discovered during the BFS.

Skip Next Forward Layer. The DFS proceeds along edges outgoing from the last forward search layer independent from the target vertex being seen only by the forward search (gray in Figure 1) or also by the backward search (orange in Figure 1). However, the former type of vertex cannot be part of a shortest s - t path. By saving the number of explored layers of the forward search we can avoid the exploration of such vertices, thus limiting the DFS to vertices colored blue, green, or orange in Figure 1. With this optimization, the combined search space during augmentation (lines 3,4 in Algorithm 1) is almost limited to the search space of the BFS. The only additional edges that are visited originate from the intersection of the forward and backward search.

4 Experimental Evaluation

In this section, we investigate the performance of our algorithm *DinitzOPT*. First, we compare it to established approaches on real-world networks in Section 4.1. We additionally examine the scaling behavior and how the comparison is affected by problem size, i.e., is there an asymptotic improvement over other algorithms? Then, Section 4.2 evaluates to which extent the different optimizations contribute to better run times and search space. In Section 4.3 we analyze the algorithms in a specific application (Gomory-Hu trees) and compare their usability beyond the speed of the actual flow computation. To this end, we test three different approaches to obtain a cut with the Push-Relabel algorithm.



■ **Figure 2** Runtime comparison of flow computations. The 20 computed flows per instance are divided into *low* and *high* terminal pairs. For *low*, the terminal degree is between 0.75 and 1.25 times the average degree. For *high*, it is between 10 and 100 times the average degree. Pairs are chosen uniformly at random from all vertices with the respective degree.

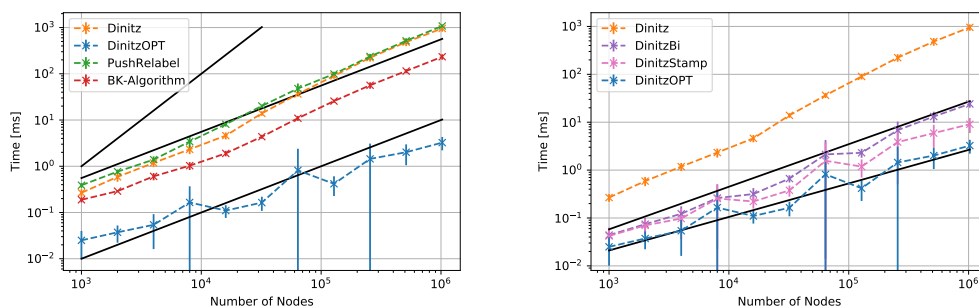
4.1 Runtime Comparison

In this section we compare our new approach to three existing algorithms: Dinitz [13], Push-Relabel [18], and the Boykov-Kolmogorov (BK) algorithm [8]. We modified their respective implementations to support our experiments. This also includes some minor performance-relevant changes listed in the long version of the paper along with further details on the datasets [5]. The experiments include two synthetic and eight real-world networks. All networks are undirected and all but *visualize-us* and *actors* are unweighted. We restrict our experiments in this section to the flow computation only excluding, e.g., loading times and resetting flow values between runs. For Push-Relabel we only measure the computation of the *preflow*, which is sufficient to determine the value of the flow/cut. Figure 2 shows the resulting run times. For this plot, the terminals were chosen uniformly at random from the set of vertices with degree close to the average (*low*) or considerably higher degree (*high*).

One can see that Dinitz and Push-Relabel display comparable times while BK is slightly slower on most large instances. DinitzOPT consistently outperforms the other algorithms by one to three orders of magnitude. The variance is also higher for DinitzOPT with *low* pairs approximately one order of magnitude faster on average than *high* pairs. This is best seen in the *girg100000* instance and suggests that DinitzOPT is able to better exploit easy problem instances. For all other algorithms the effect of the terminal degree on the run time is barely noticeable. Another observation is that all algorithms display drastically lower run times than their respective worst-case bounds would suggest.

The times in our experiments are close to what one might expect from linear algorithms. For example, Dinitz computes a flow on the *as-skitter* instance in one second. Considering the tight $O(mn^{2/3})$ bound in unweighted networks and assuming the throughput per second to be around 10^8 – which is a generous guess for graph algorithms – would result in an estimate of 30 minutes per flow. In contrast to our results, earlier studies found Dinitz to be slower than Push-Relabel and both algorithms clearly super-linear on a series of synthetic instances [1]. However, these synthetic instances exhibit specifically crafted hard structures that are placed between designated source and sink vertices. These instances thus present substantially more challenging flow problems.

Effect of the Terminal Degree. In the following, we discuss the effect of terminal degree and structure of the cut on the run time of Dinitz and DinitzOPT. Note that the terminal degree is an upper bound on the size of the cut in unweighted networks. Moreover, the



(a) Runtime scaling of flow algorithms.

(b) Scaling of Dinitz variants.

■ **Figure 3** (a) The average time per flow over multiple GIRGs and terminal pairs. (b) This plot differs from Figure 3a only in the set of displayed algorithms.

terminal degree in our experiments is based on the average degree, which is assumed to be constant in many real-world networks [3]. Thus, the $O(mC)$ bound for augmenting path based algorithms, with C being the size of the cut, implies not only a linear bound for the eight unweighted networks in our experiments, but would also explain faster *low* pairs. Surprisingly, DinitzOPT exploits low terminal degrees much more than Dinitz. Another explanation for faster *low* pairs is that many cuts are close around one terminal, which is consistent with previous observations about cuts in scale-free networks [27, 31]. Moreover, Dinitz tends to perform well when the source side of the cut is small [28]. Although this does not fully explain why DinitzOPT is more sensitive to the terminal degree, we observe in Section 4.3 that Dinitz slows down massively when the source degree is high, even with low sink degree. Since DinitzOPT always advances the side with smaller volume during bidirectional search it does not matter which terminal has the higher degree.

Scaling. We perform additional experiments to analyze the scaling behavior of the algorithms. Since real networks are scarce and fixed in size, we generate synthetic networks to gradually increase the size while keeping the relevant structural properties fixed. Geometric Inhomogeneous Random Graphs (GIRGs) [9], a generalization of Hyperbolic Random Graphs [25], are a scale-free generative network model that captures many properties of real-world networks. The efficient generator [4] allows us to benchmark our algorithms on differently sized networks with similar structure. Figure 3a and Figure 3b show the results.

We measure the run time over a series of GIRGs with the number of nodes growing exponentially from 1000 to 1 024 000 with 10 iterations each. In each iteration, we sample a new random graph with average degree 10, power-law exponent 2.8, dimension 1, and temperature 0. The run time for each algorithm is then averaged over 10 uniform random pairs of vertices with degree between 10 and 20. Standard deviation is shown as error bars. The lower half of the symmetric error bars seems longer due to the log-axis. We add a quadratic and two linear functions in Figure 3a. Figure 3b shows the functions $n^{0.88}$ and $n^{0.7}$ representing the theoretical upper bound and previously observed typical run times, respectively, for the bidirectional search on hyperbolic random graphs with the chosen power-law exponent [6].

Dinitz, Push-Relabel and BK show a near-linear running time. Compared to the linear functions in Figure 3a, Dinitz and Push-Relabel seem to scale slightly worse than linear, while DinitzOPT scales better than linear. In a construction with super-sink and super-source, a similar scaling was observed for Push-Relabel on the Yahoo Instant Messenger graph [26].

■ **Table 1** Total run times and search space of visited edges for the five intermediate versions of our Dinitz implementation during the computation of 1000 flows in **as-skitter**. Terminals are chosen like *low* pairs in Figure 2. The first seven columns show times in seconds accumulated over all flow computations. BUILD is the construction of the residual network that is reused for all flow computations, RESET means clearing flow on edges between computations, INIT includes initialization of distances and counters per round, BFS and DFS refer to the respective subroutines, FLOW is the summed time during flow computations (sum of BFS, DFS, INIT), and TOTAL is the run time of the whole application including reading the graph from file. The last three columns contain the search space relative to the number of edges in the graph in percent. Search space columns for BFS and DFS are per round, while the FLOW column lists the search space per flow, e.g., Dinitz visits on average 65.66% of all edges per BFS and every edge is visited about 5.58 times on average in one flow computation.

	BUILD	RESET	MaxFlow					Search Space [%]		
			INIT	BFS	DFS	FLOW	TOTAL	BFS	DFS	FLOW
Dinitz	0.50	56.79	14.87	405.46	426.80	847.13	904.85	65.66	63.64	558.04
DinitzBi	0.55	58.15	21.02	2.78	8.94	32.73	91.82	0.26	1.87	8.38
DinitzReset	0.50		20.73	2.47	8.01	31.20	32.06	0.26	1.87	8.38
DinitzStamp	0.55			2.51	10.30	12.81	13.72	0.26	1.87	8.38
DinitzOPT	0.55			2.40	1.06	3.46	4.22	0.26	0.20	2.03

4.2 Optimizations in Detail

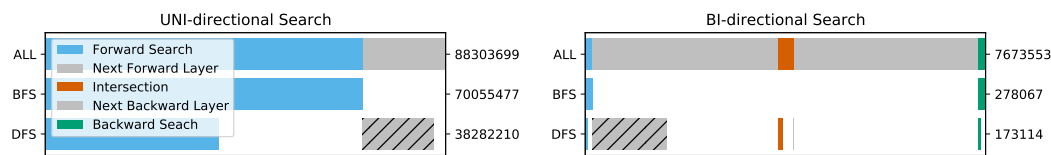
Instead of considering all combinations of optimizations, we individually add them in a specific order, such that the next change always tackles a performance bottleneck. In fact, additional benchmarks reveal that the current optimization speeds up the computation more than enabling all other remaining changes together. The four incrementally more optimized versions of the algorithm are: DinitzBi, DinitzReset, DinitzStamp, and DinitzOPT.

Experimental Setup. The experiments and benchmarks in this section consider 1000 uniform random terminal pairs close to the average degree on the **as-skitter** instance. The average distance between source and sink in the initial network is 4.2. The average number of rounds until a maximum flow is found is 4.8, where the last round runs only the BFS to verify that no augmenting path exists. Only counting rounds before the last round, 2.9 units of flow are found on average per round. Out of the 1000 cuts, 882 have value equal to the degree of the smaller terminal. Table 1 shows profiler results and search space for Dinitz and the optimized versions of the algorithm. Figure 4 compares the search space with and without bidirectional search.

Bidirectional Search. Dinitz takes 15 minutes to compute the 1000 flows and the search space per flow is more than five times the number of edges on average. Almost all of that time is spent in BFS or DFS. The bidirectional Dinitz reduces the flow-time from 14 minutes to 30 seconds, an improvement by a factor of 25.

The search space is reduced by factors of 252 for BFS, 34 for DFS, and 67 per flow. It is interesting to note, that the search space of BFS during the last round of each flow changes even more. In this round the BFS will find no s-t path. The bidirectional search visits 39 edges on average, while the normal breadth-first-search visits 44% of the graph. This not only emphasizes that the cuts are close around one terminal, but also shows that the bidirectional search heavily exploits this structure.

The run time does not fully reflect this drastic reduction in search space, because DFS and BFS no longer dominate the flow computation. The initialization time per round increased by 50%, which can be explained by the additional distance label per node to store the



■ **Figure 4** Average number of edges visited per flow computation for the terminal pairs used in Table 1, partitioned as in Figure 1. *Forward/Backward Search* represent the edges explored by the respective search. *Next Forward/Backward Layer* denote the edges that would be explored in the next step of the BFS. Edges in the *Intersection* originate from vertices in both upcoming BFS-layers. The BFS and DFS bars show the edges that are actually visited by the algorithm. The shaded area indicates the edges skipped by our last optimization (from DinitzStamp to DinitzOPT in Table 1) and is excluded in the sum on the right.

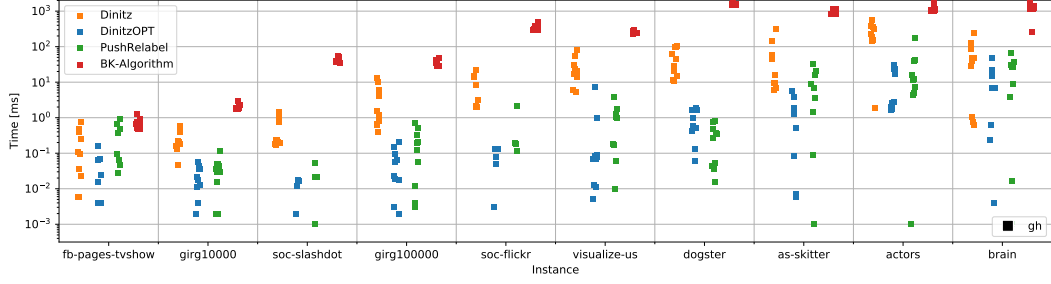
distance to the sink (now 3 ints instead of 2). Although the initialization is a simple linear operation in the number of nodes, it takes twice as long as BFS and DFS combined. The real bottleneck, however, is to reset the flow values between computations. RESET takes almost a full minute which is twice as long as computing the flows.

Reset flow between computations. Between flow computations, the residual capacity of all edges has to be reset before another flow can be found. After changing the BFS to a bidirectional search, resetting the flow on all edges between computations dominates the run time. To reduce the time of our benchmarks, and to make the code more efficient in situations where multiple flows are computed in the same network, we address this bottleneck. Instead of explicitly resetting flow values for all edges, we remember the edges that contain flow and reset only those. This change is not mentioned in Section 3 because it does not speed up a single flow computation.

This change reduces the time for RESET to the point that it is no longer detected by the profiler, while other operations are not affected. The total time to compute all 1000 flows is thus three times lower with the flow computation making up for almost all spent time. The slowest part of the flow computation itself is still the initialization with 21 of the 31 seconds.

Time Stamps. The distance labels and counters per node are initialized each round. Using time stamps eliminates the need for initialization while adding a small overhead to DFS. The flow computation gets 2.4 times faster with 13 seconds instead of 31. After introducing the time stamps, the DFS is the new bottleneck and makes up for about 80% of flow time.

Skip Next Forward Layer. This change prevents the DFS from visiting vertices beyond the last layer of the forward search that are not also seen by the backwards search. In Figure 4 the skipped part is shaded. This optimization reduces the average search space for DFS during one round from almost 2% of all edges to just 0.2%. The improvement in search space is reflected by the profiler results. DFS is sped up from 10 seconds to just one second, which is faster than the BFS. The resulting time to compute all 1000 flows is 3.46 seconds, which is only 7 times slower than building the adjacency list in the beginning. In total, the time to compute the flows with the optimized Dinitz is 245 times faster than the unmodified Dinitz.



■ **Figure 5** Runtime comparison of flow computations. The 10 terminal pairs per instance are uniformly chosen out of the $n - 1$ cuts required by Gusfield’s algorithm.

4.3 Gomory-Hu Trees

In the last sections we observed that heterogeneous network structure yields easy flow problems that can be solved significantly faster than the construction of the adjacency list. This performance becomes important in applications that require multiple flows to be found in the same network. Gomory-Hu trees [20] fit this setting and have applications in graph clustering [15]. A Gomory-Hu tree (GH-tree) of a network is a weighted tree on the same set of vertices that preserves minimum cuts, i.e., each minimum cut between any two vertices s and t in the tree is also a minimum s - t cut in the original network. Thus, they compactly represent s - t cuts for all vertex pairs of a graph. For the construction of a GH-tree, we use Gusfield’s algorithm [21] that requires $n - 1$ cut-oracle calls in the original graph.

In this section we evaluate the performance of max-flow algorithms for the construction of Gomory-Hu trees in heterogeneous networks. We will see that the terminal pairs required for Gusfield’s algorithm yield easier flow problems than uniform random pairs. DinitzOPT is able to make use of this easy structure to achieve surprisingly low run times, so is Push-Relabel when only considering the computation of the flow value. However, we find that the need to extract the source side of the cut hinders Push-Relabel to benefit from this performance.

Flow Computation on Gusfield Pairs. Figure 5 shows the same networks and algorithms as in Figure 2 but with terminal pairs sampled out of the $n - 1$ flow computations needed by Gusfield’s algorithm. The run times for all algorithms except the BK-Algorithm have high variance and are spread over up to four orders of magnitude for the larger instances. Although results for different terminal pairs vary greatly, BK seems to be the slowest algorithm followed by Dinitz. DinitzOPT and PR have comparable but significantly lower run times than the other algorithms. For example, 6 out of the 10 *gh* pairs measured for the **soc-slashdot** instance are solved by DinitzOPT and Push-Relabel faster than one microsecond which is the precision of our measurements. This suggests, that these algorithms are more sensitive to the varying difficulty of the flow computations for *gh* pairs. Our speedup over the Push-Relabel algorithm on *gh* pairs is not as pronounced as for the random pairs in Section 4.1. On the **dogster** instance PR is even faster than DinitzOPT.

To further investigate why *gh* pairs are this easy to solve, we analyze a complete run of all pairs needed by Gusfield’s algorithm on the **soc-slashdot** instance. In Gusfield’s algorithm each vertex is the source once, thus the average degree of the source is the average degree of the graph (10.24). In contrast, the average degree of the sink is ca. 1500, which hinders the benefit of bidirectional search. Uni-directional Dinitz slows down by a factor of 15 when computing the flows with switched terminals. The average distance between two vertices

in the original network is 4.16, but interestingly here the average distance from source to sink is 1.78. Out of the 70k flow computations, 56k are trivial cuts around one terminal. Computing a flow for a single s-t pair takes 2.76 rounds on average with the last round only to confirm that the flow is optimal.

DinitzOPT and Push-Relabel are both extremely fast on *gh* pairs. DinitzOPT takes 2.5 seconds to compute all $n = 70k$ required flows, while PR needs 5 seconds. To obtain the 5 seconds for PR we exclusively measured the preflow computation, but PR is not limited by the time to compute the preflow. Actually, the entire computation of the Gomory-Hu tree on the `soc-slashdot` instance takes 12 minutes with Push-Relabel and 2.6 seconds with DinitzOPT. Instead of being caused by the Gusfield logic – which actually makes up less than 3% of the run time when using DinitzOPT as oracle – the bottleneck when using PR as a cut oracle is not the flow computation, but initialization and extracting the cut. The drastic difference in run time is in part due to the optimizations we added to DinitzOPT to reduce time between flow computations, while the Push-Relabel implementation recreates the auxiliary data structures, except the adjacency list, before each flow. However, in the following we will see that a large amount of Push-Relabels run time is actually necessary to extract the cuts for Gusfield’s algorithm.

Computing Cuts with Push-Relabel. In Gusfield’s algorithm we have to iterate over all vertices in the source-side of the cut. For Dinitz algorithm we can obtain a cut by doing a BFS from the source. However, the PR algorithm only computes a preflow. We outline the following three approaches to extract the cut and show that each has major drawbacks.

Convert. Compute a preflow, convert it into a flow, then run BFS from the source.

T-Side. Compute a preflow, run BFS backwards from the sink, then take complement.

Swap. Compute a preflow from sink to source, then run BFS backwards from the source.

The most straightforward way to get a cut from a preflow is to convert it into a flow. Then, as for Dinitz, one partition of a min-cut can be identified by reachability from the source in the residual network. In previous works, the conversion from preflow to flow makes up only a small fraction of the running time [11, 12]. For Gusfield pairs, however, Figure 6 shows that the conversion highly dominates the computation of the preflow. Only about 5 seconds of the 12 minutes of the complete run are spent in preflow computation.

To circumvent the conversion, we use the observation that one can obtain a cut directly from the preflow by finding all sink-reaching vertices in the residual network. Since Gusfield requires the source-side of the cut, the complement of the found set of vertices can be used. Unfortunately, doing the backward search from the sink is even more expensive than the conversion. An explanation for this is the large sink side of the cut. Using this *T-side* approach to identify the cut for DinitzOPT takes 4.5 minutes which is a factor 100 slower than identifying the cut via the source-side for DinitzOPT.

Making use of the fact that the source side of the cut is much smaller than the sink side, the drawbacks of the previous approach can be avoided in undirected networks by computing the preflow from sink to source. A cut can then be extracted by determining the vertices that can reach the original source in the residual network. The drawback of this method is that the preflow computation slows down massively from 5s to 47 minutes.

In conclusion, the *convert* approach is the fastest with just above 12 minutes followed by *T-side* with 18 minutes and *swap* with almost an hour. However, all three methods perform significantly worse than DinitzOPT, not because PR flow computations are slow, but both methods to avoid the four minutes run time of preflow-conversion imply even worse performance cost; either due to a breadth-first search that has to traverse almost the whole graph (*T-side*) or due to significantly slower preflow computations (*Swap*).



Figure 6 Distribution of spent time during Gusfield’s algorithm on the `soc-slashdot` instance with three approaches to use the Push-Relabel algorithm as a min-cut oracle. We split the measurements into initialization, preflow, conversion, and cut identification. The time overhead for measurement, logging, and the logic of Gusfield’s algorithm is included in the numbers on the right but excluded in the bars.

5 Conclusion

We presented a modified version of Dinitz’s algorithm with greatly improved run time and search space on real-world and generated scale-free networks. The scaling behavior appears to be sublinear, which matches previous theoretical and empirical observations about the running time of balanced bidirectional search in scale-free random networks. While these theoretical bounds apply during the first round of our algorithm, it is still unknown whether the analysis can be extended to account for the changes in the residual network. Our experiments, however, indicate that the search space remains small in subsequent rounds.

We observe that the low diameter and heterogeneous degree distribution lead to small and unbalanced cuts that our algorithm finds very efficiently. The flow computations required to compute a Gomory-Hu tree are even easier, making usually insignificant parts of the tested algorithms be a bottleneck. For example, the preflow conversion leads to Push-Relabel being greatly outperformed by our algorithm in this setting.

References

- 1 Ravindra K. Ahuja, Murali Kodialam, Ajay K. Mishra, and James B. Orlin. Computational investigations of maximum flow algorithms. *European Journal of Operational Research*, 97(3):509–542, 1997. doi:10.1016/S0377-2217(96)00269-X.
- 2 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: Theory, algorithms and applications*. Prentice-Hall, Inc., 1993.
- 3 Albert-László Barabási. *Network science*. Cambridge university press, 2016.
- 4 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.21.
- 5 Thomas Bläsius, Tobias Friedrich, and Christopher Weyand. Efficiently computing maximum flows in scale-free networks. *CoRR*, abs/2009.09678, 2020. arXiv:2009.09678.
- 6 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 *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.20.

- 7 Michele Borassi and Emanuele Natale. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.20.
- 8 Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. doi:10.1109/TPAMI.2004.60.
- 9 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.
- 10 Bala G. Chandran and Dorit S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research*, 57(2):358–376, 2009.
- 11 B. V. Cherkassky and A. V. Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997. doi:10.1007/pl00009180.
- 12 U. Derigs and W. Meier. Implementing Goldberg’s max-flow-algorithm — A computational investigation. *Zeitschrift für Operations Research*, 33(6):383–403, 1989. doi:10.1007/BF01415937.
- 13 Yefim Dinitz. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.
- 14 Shimon Even and R. Endre Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, 1975. doi:10.1137/0204043.
- 15 Gary William Flake, Robert E. Tarjan, and Kostas Tsioutsoulis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004. doi:10.1080/15427951.2004.10129093.
- 16 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.
- 17 Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Robert E. Tarjan, and Renato F. Werneck. Maximum Flows by Incremental Breadth-First Search. In *19th Annual European Symposium on Algorithms (ESA 2011)*, Lecture Notes in Computer Science, pages 457–468. Springer, 2011. doi:10.1007/978-3-642-23719-5_39.
- 18 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.
- 19 Andrew V. Goldberg and Robert E. Tarjan. Efficient maximum flow algorithms. *Commun. ACM*, 57(8):82–89, 2014. doi:10.1145/2628036.
- 20 R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
- 21 Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990. doi:10.1137/0219009.
- 22 Felix Halim, Roland H.C. Yap, and Yongzheng Wu. A MapReduce-Based Maximum-Flow Algorithm for Large Small-World Network Graphs. In *2011 31st International Conference on Distributed Computing Systems*, pages 192–202, 2011. ISSN: 1063-6927. doi:10.1109/ICDCS.2011.62.
- 23 Dorit S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56(4):992–1009, August 2008. doi:10.1287/opre.1080.0524.
- 24 Alexander V. Karzanov. On finding a maximum flow in a network with special structure and some applications. *Matematicheskie Voprosy Upravleniya Proizvodstvom*, 5:81–94, 1973.
- 25 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3), 2010. doi:10.1103/physreve.82.036106.
- 26 Kevin Lang. Finding good nearly balanced cuts in power law graphs. Technical Report YRL-2004-036, Yahoo! Research Labs, 2004.

- 27 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009. doi:10.1080/15427951.2009.10129177.
- 28 Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.94.
- 29 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.
- 30 Boris Schäling. *The boost C++ libraries*. Boris Schäling, 2011. URL: <https://theboostcpplibraries.com/>.
- 31 S.-W. Son, H. Jeong, and J. D. Noh. Random field ising model and community structure in complex networks. *The European Physical Journal B*, 50(3):431–437, 2006. doi:10.1140/epjb/e2006-00155-4.
- 32 Tanmay Verma and Dhruv Batra. MaxFlow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *Proceedings of the British Machine Vision Conference 2012*. British Machine Vision Association, 2012. doi:10.5244/c.26.61.

Asymptotically Optimal Welfare of Posted Pricing for Multiple Items with MHR Distributions

Alexander Braun ✉

Institute of Computer Science, Universität Bonn, Germany

Matthias Buttkus

Institute of Computer Science, Universität Bonn, Germany

Thomas Kesselheim ✉

Institute of Computer Science, Universität Bonn, Germany

Abstract

We consider the problem of posting prices for unit-demand buyers if all n buyers have identically distributed valuations drawn from a distribution with monotone hazard rate. We show that even with multiple items asymptotically optimal welfare can be guaranteed.

Our main results apply to the case that either a buyer's value for different items are independent or that they are perfectly correlated. We give mechanisms using dynamic prices that obtain a $1 - \Theta\left(\frac{1}{\log n}\right)$ -fraction of the optimal social welfare in expectation. Furthermore, we devise mechanisms that only use static item prices and are $1 - \Theta\left(\frac{\log \log \log n}{\log n}\right)$ -competitive compared to the optimal social welfare. As we show, both guarantees are asymptotically optimal, even for a single item and exponential distributions.

2012 ACM Subject Classification Theory of computation \rightarrow Algorithmic mechanism design; Theory of computation \rightarrow Online algorithms

Keywords and phrases Prophet Inequalities, Monotone Hazard Rate, Competitive Analysis, Posted Prices, Combinatorial Auctions, Matching

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.22

Related Version *Full Version*: <https://arxiv.org/abs/2107.00526> [5]

Acknowledgements We thank the anonymous reviewers for helpful comments on improving the presentation of the paper.

1 Introduction

Posting prices is a very simple way to de-centralize markets. One assumes that buyers arrive sequentially. Whenever one of them arrives, a mechanism offers a menu of items at suitably defined prices. The buyer then decides to accept any offer, depending on what maximizes her own utility. Such a mechanism is incentive compatible by design, usually easy to explain and can be implemented online. For this reason, there is a large interest in understanding what social welfare and revenue can be guaranteed in comparison to mechanisms that optimize the respective objective.

Let us consider the following setting: There is a set of m heterogeneous items M , each of which we would like to be allocated to one of n buyers. Each buyer i has a private valuation function $v_i: 2^M \rightarrow \mathbb{R}_{\geq 0}$. We assume that valuation functions are unit-demand. That is, $v_i(S) = \max_{j \in S} v_i(\{j\})$, meaning that the value a buyer associates to a set is simply the one of the most valuable item in this set. Let $\text{SW}_{\text{opt}} = \max_{\text{allocations}} (S_1, \dots, S_n) \sum_{i=1}^n v_i(S_i)$ denote the optimal (offline/ex-post) social welfare. Note that this optimal solution is nothing but the maximum-weight matching in a bipartite graph in which all buyers and items correspond to a vertex each and an edge between the vertices of buyer i and item j has weight $v_i(\{j\})$.



© Alexander Braun, Matthias Buttkus, and Thomas Kesselheim;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 22; pp. 22:1–22:16



Leibniz International Proceedings in Informatics

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

To capture the pricing setting, we assume that the functions v_1, \dots, v_n are unknown a priori; all of them are drawn independently from the same, publicly known distribution. For every item, one can either set a static item price or change the prices dynamically over time. Buyers arrive one-by-one and each of them chooses the set of items that maximizes her utility given the current prices among the remaining items. Static prices have the advantage that they are easier to explain and thus give easier mechanisms. However, dynamic prices can yield both higher welfare and revenue because they can be adapted to the remaining supply and the remaining number of buyers to appear.

Coming back to the interpretation of a bipartite matching problem, a posted-prices mechanism corresponds to an online algorithm, where the buyers correspond to online vertices and the items correspond to offline vertices. However, not every online algorithm necessarily corresponds to a posted-prices mechanism: There might not be item prices such that the choices of the algorithm correspond to the ones by a buyer maximizing their utility.

We would like to understand which fraction of the optimal (offline/ex-post) welfare posted-prices mechanisms can guarantee. The case of a single item is well understood via *prophet inequalities* from optimal stopping theory. Let us call a posted-prices mechanism β -competitive (with respect to social welfare) if its expected welfare $\mathbf{E}[\text{SW}_{\text{pp}}]$ is at least $\beta \mathbf{E}[\text{SW}_{\text{opt}}]$. For a static price and a single item, the best such guarantee is $\beta = 1 - \frac{1}{e} \approx 0.63$ [11, 15]; for dynamic pricing and a single item, it is $\beta \approx 0.745$ [1, 11]. There are a number of extensions of these results to multiple items (see Section 1.3 for details), also going beyond unit-demand valuations, many of which are $O(1)$ -competitive.

The competitive ratios of $\beta = 1 - \frac{1}{e} \approx 0.63$ and $\beta \approx 0.745$ are optimal in the sense that there are distributions and choices of n such that no better guarantee can be obtained. Importantly, they are still tight when imposing a lower bound on n . That is, even for large n , there is a distribution such that if all values are drawn from this distribution the respective bound cannot be beaten.

1.1 Distributions with Monotone Hazard Rate

In this paper, we strengthen previous results by restricting the class of distributions to ones with monotone hazard rate. The single-item case is defined as follows. Consider a probability distribution on the reals with probability density function (PDF) f and cumulative distribution function (CDF) F , its hazard rate h is defined by $h(x) = \frac{f(x)}{1-F(x)}$ for x with $F(x) < 1$. It has a *monotone hazard rate* (MHR) – more precisely, increasing hazard rate – if h is a non-decreasing function. It has become a common and well-studied approach to model buyer preferences by MHR distributions. One of the reasons is that many standard distributions exhibit a monotone hazard rate such as, for example, uniform, normal, exponential and logistic distributions. (For a much more extensive list see [25].) Furthermore, the monotone hazard rate of distributions is also preserved under certain operations; for example, order statistics of MHR distributions also have an MHR distribution. Additionally, every MHR distribution is regular in the sense that its virtual value function [8, 24] is increasing.

We generalize results to multiple items and consider two fundamental settings. On the one hand, we consider *independent item valuations*, i.e. $v_{i,j} \sim \mathcal{D}_j$ is an independent draw from a distribution \mathcal{D}_j . In other words, the value of item j is independent of the value of item j' and both values are drawn from (possibly different) MHR distribution as defined above. On the other hand, we assume correlated values for items via the notion of *separable item valuations*, which are common in ad auctions [14, 27]: Each buyer has a type $t_i \geq 0$ and each item has an item-dependent multiplier α_j where now $v_{i,j} = \alpha_j \cdot t_i$. Again, $t_i \sim \mathcal{D}$ and \mathcal{D} is a distribution with monotone hazard rate. We note that this case subsumes and extends the case of k identical items.

As we show, in these cases, asymptotically optimal welfare can be guaranteed. That is, if n grows large, the social welfare when suitably choosing prices is within a $1 - o(1)$ factor of the optimum, where the $o(1)$ term is independent of the distribution as long as its marginals satisfy the MHR property. Stated differently, there is a sequence $(\beta_n)_{n \in \mathbb{N}}$ with $\beta_n \rightarrow 1$ for $n \rightarrow \infty$ such that for every number of buyers n there exists a posted-prices mechanism that takes any distribution with MHR as input and guarantees $\mathbf{E}[\text{SW}_{\text{pp}}] \geq \beta_n \mathbf{E}[\text{SW}_{\text{opt}}]$. As pointed out before, such a result does not hold for arbitrary, non-MHR distributions. Even with a single item, the limit is then upper-bounded by ≈ 0.745 .

A similar effect has already been observed by Giannakopoulos and Zhu [17]. They show that the revenue of static pricing for a single item with MHR distributions asymptotically reaches the optimal revenue. In contrast, our results concern welfare. Still, some of our results also have implications for revenue, either because we bound the revenue or because one could apply the results to virtual values.

1.2 Our Results and Techniques

We design mechanisms for both independent and separable item valuations. The ones using dynamic prices ensure a $\left(1 - O\left(\frac{1}{\log n}\right)\right)$ -fraction of the expected optimal social welfare. The ones using static prices guarantee a $\left(1 - O\left(\frac{\log \log \log n}{\log n}\right)\right)$ -fraction. We also show that these guarantees are best possible, even in the case of only a single item. Note that the bounds are independent of the number of items m , which may also grow in the number of buyers n .

Independent Valuations (Section 3)

The technically most interesting result is the one on dynamic pricing when values are independent across items. The idea is to set prices so that the offline optimum is mimicked. If item j is allocated in the optimal allocation with probability q_j , then we would like it to be sold in every step with an ex-ante probability of $\frac{q_j}{n}$. However, analyzing such a selling process is still difficult because items are incomparable and bounds for MHR distributions cannot be applied directly to draws from multiple distributions, which are not necessarily identical. To bypass this problem, we introduce a reduction that allows us to view item valuations not only as independent but also as identically distributed. To this end, we compare the selling process of our mechanism to a hypothetical setting, in which buyers do not make their decisions based on the actual utility but in quantile space. We observe that the revenue of both is identically distributed and utility is maximized in the former mechanism. As a consequence, the welfare obtained by the quantile allocation rule is a feasible lower bound on the welfare of the sequential posted-prices mechanism. Only afterwards, we can apply a concentration bound due to the MHR restriction.

The idea of our mechanism using static prices is to set prices suitably high in order to bound the revenue of our mechanism with a sufficient fraction of the optimum. While all other bounds apply for any number of items m , this bound unfortunately requires $m \leq \frac{n}{(\log \log n)^2}$. We leave it as an open problem to extend the result for larger number of items.

Separable Valuations (Section 4)

Our way of setting dynamic prices in the case of separable valuations is similar to the approach in independent valuations. This setting is even a little simpler because we can assume without loss of generality that there are as many items as buyers. Our pricing strategy ensures that in each step each item is sold equally likely as well as one item is sold

for sure. In the analysis, we observe that we match a buyer and an item if the quantile of the buyer's value is in a specific range. Now, the MHR property comes into play which allows to bound quantiles of the distribution in a suitable way.

In the static case, to lower-bound the welfare of our mechanism, we compare it to the one of the VCG mechanism [10, 18, 28] which maximizes social welfare. To this end, we split social welfare in revenue and utility and bound each quantity separately. That is, we relate the revenue and the sum of buyer utilities of our posted-pricing mechanism to the ones of the VCG mechanism. For the revenue, we set prices fairly low to ensure that we sell all items with reasonably high probability. Still, these prices are high enough to use the MHR property and derive a suitable lower bound of the prices. The utility comparison is more complicated, we solve this issue by an unusual application of the equality of expected revenue and virtual welfare due to Myerson [24].

Optimality (Section 5)

The achieved bounds on the competitive ratio are optimal for both dynamic as well as static pricing. We show this by considering the single-item case with an exponential distribution, which is a special case of both independent and separable valuations. For dynamic prices, we use the correspondence to a Markov decision process showing that no online algorithm is better than $1 - \Omega\left(\frac{1}{\log n}\right)$ -competitive. Then we also show that the competitive ratio cannot be better than $1 - \Omega\left(\frac{\log \log \log n}{\log n}\right)$ for any choice of a static price by writing out the expected social welfare explicitly.

Subadditive Valuations (Section 6)

We also demonstrate that our techniques are applicable beyond unit-demand settings by giving mechanisms for the more general class of subadditive valuation functions. Our dynamic pricing mechanism is $1 - O\left(\frac{1 + \log m}{\log n}\right)$ -competitive for subadditive buyers. We complement this by a static pricing mechanism which is $1 - O\left(\frac{\log \log \log n}{\log n} + \frac{\log m}{\log n}\right)$ -competitive. Both guarantees can be derived by showing that the revenue of the posted pricing mechanism is at least as high as the respective fraction of the optimal social welfare. As a consequence, these bounds directly imply the competitive ratios for welfare and revenue. For small m , these bounds are again tight by our optimality results. Obtaining tight bounds for large m still remains an open problem.

1.3 Further Related Work

As mentioned already, our setup restricted to a single item is highly related to prophet inequalities. Prophet inequalities have their origin in optimal stopping theory, dating back to the 1970s [22]. Only much later they were considered as a tool to understand the loss by posting prices as opposed to using other mechanisms. In this context, Samuel-Cahn's result [26] then got the interesting interpretation that posting an appropriately chosen static price for an item is $\frac{1}{2}$ -competitive for any buyer distributions; different buyers may even be drawn from different distributions. This guarantee is optimal, even for dynamic pricing.

Improvements for the single-item case are only possible by imposing further assumptions. Most importantly, this concerns the case that all buyer values are drawn from the same distribution. While already discussed by Hill et al. [20], this problem has been solved only very recently by devising an ≈ 0.745 -competitive mechanism that relies on dynamic pricing [1, 11]. By using static pricing, one cannot be better than $1 - \frac{1}{e} \approx 0.63$ -competitive [11, 15].

Better guarantees can also be achieved by assuming that multiple, identical items are for sale. In this case, one can use concentration results. The respective competitive ratios tend to 1 for a growing number of item copies. Hajiaghayi et al. [19] gave the first guarantee for such a setting, Alaei [2] later improved it to tightness.

For identical regular distributions, all of the above results also apply to welfare as well as revenue maximization because the prices can be imposed in the space of the virtual valuation [24]; also see the work by Chawla et al. [9]. Also impossibility results transfer [12].

When it comes to multiple, heterogeneous items, there is a significant difference between welfare and revenue maximization because Myerson's characterization does not apply anymore. For welfare maximization, Feldman et al. [16] show that static item prices still yield a competitive ratio of $\frac{1}{2}$ even for XOS valuations and not necessarily identically distributed buyer valuations. Concerning subadditive valuation functions, Zhang [29] gives a $O(\log m / \log \log m)$ -competitive prophet inequality, Dütting et al. [13] show how to obtain a competitive ratio of $O(\log \log m)$. The only improvement for identically distributed buyers is to $1 - \frac{1}{e}$ for unit-demand buyers based on dynamic pricing [15]. Among others, Chawla et al. [7] considered a combinatorial generalization of such a setting with many item copies (see Lucier's survey [23] on a broader overview of combinatorial generalizations).

For revenue maximization, one usually imposes the additional assumption that items are independent. This makes it possible to also apply prophet inequalities on the sequence of items rather than buyers and thus maximize revenue for unit-demand buyers via posted prices [8, 9]. Cai and Zhao [6] consider more general XOS and subadditive valuations and apply a duality framework instead. They design a posted-prices mechanism with an entry fee that gives an $O(1)$ or $O(\log m)$ approximation to the optimal revenue. In Dütting et al. [13], the approximation of the optimal revenue for subadditive valuations is improved to $O(\log \log m)$.

There are surprisingly few results on pricing and prophet inequalities that derive better guarantees by imposing additional assumptions on the distribution. Babaioff et al. [4] consider the problem of maximizing revenue when selling a single item to one of n buyers drawn i.i.d. from an *unknown* MHR distribution with a bounded support $[1, h]$. If n is large enough compared to h , they get a constant-factor approximation to the optimal revenue using dynamic posted prices. Note that in contrast, in our paper, we assume to know the underlying distributions perfectly. Giannakopoulos and Zhu [17] consider revenue maximization in the single-item setting with valuations drawn independently from the same MHR distribution. They show that by offering the item for the same static price to all bidders one can achieve asymptotically optimal revenue. More precisely, one of their main results is that one gets within a factor of $1 - O\left(\frac{\ln \ln n}{\ln n}\right)$. While they claim this result is “essentially tight”, we show that the best factor is indeed $1 - \Theta\left(\frac{\ln \ln \ln n}{\ln n}\right)$ because it is a special case of our results (see Section 6). It is not clear, how one could apply their result to welfare maximization as the MHR property is not preserved when moving between virtual and actual values. Furthermore, their results do not admit any apparent generalization to multiple items. Jin et al. [21] also consider revenue maximization in the single-item setting with identical and independent MHR values but in a non-asymptotic sense, providing a bound for every n .

2 Preliminaries

We consider a setting of n buyers N and a set M of m items. Every buyer has a valuation function $v_i: 2^M \rightarrow \mathbb{R}_{\geq 0}$ mapping each bundle of items to the buyer's valuation. We assume buyers to be unit-demand, that is $v_i(S) = \max_{j \in S} v_{i,j}$. The functions v_1, \dots, v_n are unknown

a priori but all drawn independently from the same, publicly known distribution \mathcal{D} . Let \mathcal{D}_j be the marginal distribution of $v_{i,j}$, which is the value of a buyer for being allocated item j . We assume that \mathcal{D}_j is a continuous, real, non-negative distribution with monotone hazard rate. That is, let F_j be the cumulative distribution function of \mathcal{D}_j and f_j its probability density function. The distribution's hazard rate is defined as $h_j(x) = f_j(x)/(1 - F_j(x))$ for all x such that $F_j(x) < 1$. We assume a *monotone hazard rate*, which means that h_j is a non-decreasing function. Equivalently, we can require $x \mapsto \log(1 - F_j(x))$ to be a concave function.

We design *posted-prices mechanisms*. That is, the buyers arrive one by one in order $1, \dots, n$. In the i -th step, buyer i arrives and has the choice between all items which have not been allocated so far. Let $M^{(i)}$ denote this set of available items. The mechanism presents the i -th buyer a menu of prices $p_j^{(i)}$ for all items $j \in M^{(i)}$. The buyer then picks the item $j_i \in M^{(i)}$ which maximizes her *utility* $v_{i,j_i} - p_{j_i}^{(i)}$ if positive¹. Buyer i and item j_i are matched immediately and irrevocably. If buyer i has negative utility for all items $j \in M^{(i)}$, then buyer i does not buy any item and remains unmatched. Generally, the prices for buyer i may depend arbitrarily on $M^{(i)}$ and the distribution \mathcal{D} . We call prices *static* if there are p_1, \dots, p_m such that $p_j^{(i)} = p_j$ for all i and all j .

Fix any posted-prices mechanism and let j_i denote the item allocated to buyer i (set $j_i = \perp$ if i remains unmatched in the mechanism). The *expected social welfare* of the mechanism is given by $\mathbf{E}[\sum_{i=1}^n v_{i,j_i}] =: \mathbf{E}[\text{SW}_{\text{pp}}]$. In comparison, let the social welfare maximizing allocation assign item j_i^* to buyer i . Its expected social welfare is therefore given by $\mathbf{E}[\sum_{i=1}^n v_{i,j_i^*}] =: \mathbf{E}[\text{SW}_{\text{opt}}]$.

We call a posted-prices mechanism β -*competitive* if it ensures that the expected social welfare of its allocation is at least a β -fraction of the expected optimal social welfare. That is, for any choice of distribution,

$$\mathbf{E}[\text{SW}_{\text{pp}}] = \mathbf{E}\left[\sum_{i=1}^n v_{i,j_i}\right] \geq \beta \mathbf{E}\left[\sum_{i=1}^n v_{i,j_i^*}\right] = \beta \mathbf{E}[\text{SW}_{\text{opt}}] \quad .$$

3 Asymptotically Tight Bounds for Independent Valuations

In this section, we show how to derive bounds if the buyers' values are independent across items. That is, each $v_{i,j} \sim \mathcal{D}_j$ is drawn independently from a distribution with monotone hazard rate. This is a standard assumption when considering multiple items [8, 9]. As a consequence, the distribution over valuations is a product distribution $v_i = (v_{i,1}, \dots, v_{i,m}) \sim \mathcal{D} = \prod_{j=1}^m \mathcal{D}_j$ for any $i \in N$ and every \mathcal{D}_j satisfies the MHR condition.

3.1 Dynamic prices

We first consider the case of dynamic pricing mechanisms. Without loss of generality, we can assume that $m \geq n$. If we have less items than buyers, i.e. $m < n$, we can add dummy items with value 0 to ensure $m = n$. Matching i to one of these dummy items in the mechanism then corresponds to leaving i unmatched. Observe that technically a point mass on 0 is not a MHR distribution. However, all relevant statements still apply.

¹ We can assume that any buyer is buying at most one item as buyers are unit-demand. Hence, no buyer can increase utility by buying a second (lower valued) item.

Our mechanism is based on a pricing rule which balances the probability of selling a specific item. To this end, let $M^{(i)}$ be the set of remaining items as buyer i arrives. We determine dynamic prices such that one item is allocated for sure in every step. Therefore, always $|M^{(i)}| = m - i + 1$. We can now define $q_j^{(i)}$ to be the probability that item j is allocated in the “remaining” offline optimum on $M^{(i)}$ and $n - i + 1$ buyers if $j \in M^{(i)}$ and 0 else. In other words, if $j \in M^{(i)}$, $q_j^{(i)}$ is the probability that item j is allocated in the offline optimum constrained to buyers $1, \dots, i - 1$ receiving the items from $M \setminus M^{(i)}$. The prices $(p_j^{(i)})_{j \in M^{(i)}}$ are now chosen such that buyer i buys item j with probability $\frac{q_j^{(i)}}{n-i+1}$ and one item is allocated for sure. To see that such prices exist, observe the following: fix any price vector $\mathbf{x} = (x_j)_{j \in M^{(i)}}$ and denote by $r_j^{(i)}(\mathbf{x}) = \Pr[i \text{ buys item } j \text{ at prices } \mathbf{x} \mid M^{(i)}]$. As the random variables $v_{i,j}$ are continuous and independent, the probability that buyer i buys item j at prices \mathbf{x} given the current set of items $M^{(i)}$ is continuous in x_j . Hence, we can consider the mapping $(\phi^{(i)}(\mathbf{x}))_j = \frac{n-i+1}{q_j^{(i)}} \cdot r_j^{(i)}(\mathbf{x}) \cdot x_j$ for any $j \in M^{(i)}$ which is also continuous and hence, by the use of Brouwer’s fixed point theorem² has our desired price vector $(p_j^{(i)})_{j \in M^{(i)}}$ as fixed point. This allows us to state the following theorem.

► **Theorem 1.** *The posted-prices mechanism with dynamic prices and independent item-valuations is $1 - O\left(\frac{1}{\log n}\right)$ -competitive with respect to social welfare.*

Note that in the case $m \leq n$ we will always have $q_j^{(i)} = 1$ for $j \in M^{(i)}$. This significantly simplifies the argument. The proof for the general case can be found in the full version of the paper. Here, we give a sketch with the major steps and key techniques.

In order to bound the social welfare obtained by the posted-prices mechanism, we consider the following *quantile allocation rule*. For any $j \in M^{(i)}$ with $q_j^{(i)} > 0$, compute $R_j^{(i)} := F_j(v_{i,j})^{\frac{1}{q_j^{(i)}}}$ and allocate buyer i the item j which maximizes $R_j^{(i)}$. Observe that by this definition for any i , any j and any $t \in [0, 1]$,

$$\Pr[R_j^{(i)} \leq t] = \Pr[F_j(v_{i,j}) \leq t^{q_j^{(i)}}] = t^{q_j^{(i)}},$$

as $F_j(v_{i,j}) \sim \text{Unif}[0, 1]$. In particular, note that for $q_j^{(i)} = 1$, this is exactly the CDF of a random variable drawn from $\text{Unif}[0, 1]$. Define indicator variables $X_{i,j}$ which are 1 if buyer i is allocated item j in the quantile allocation rule. Then, we can observe the following.

► **Observation 2.** *It holds*

$$\Pr[X_{i,j} = 1 \mid M^{(i)}] = \frac{q_j^{(i)}}{n-i+1}.$$

Note that by this, the probability of allocating item j in step i via the quantile allocation rule is $\frac{q_j^{(i)}}{n-i+1}$, exactly as in the posted-prices mechanism.

Proof. We allocate item j in the quantile allocation rule if $R_j^{(i)} \geq R_{j'}^{(i)}$ for any $j' \in M^{(i)}$. For fixed $M^{(i)}$, also the values of $q_j^{(i)}$ are fixed. Hence, we can use independence of the $v_{i,j}$

² In addition, we can use that prices $\mathbf{x} = (x_j)_{j \in M^{(i)}}$ are always bounded by $0 \leq x_j \leq F_j^{-1}\left(1 - \frac{q_j^{(i)}}{n-i+1}\right)$ to get a convex and compact set of price vectors.

variables to compute:

$$\begin{aligned}
\Pr[X_{i,j} = 1 \mid M^{(i)}] &= \Pr\left[\max_{j' \neq j} R_{j'}^{(i)} \leq R_j^{(i)} \mid M^{(i)}\right] \\
&= \int_0^1 \Pr\left[\max_{j' \neq j} R_{j'}^{(i)} \leq t \mid M^{(i)}\right] q_j^{(i)} t^{q_j^{(i)}-1} dt \\
&= \int_0^1 \prod_{j' \neq j} \Pr[R_{j'}^{(i)} \leq t \mid M^{(i)}] q_j^{(i)} t^{q_j^{(i)}-1} dt \\
&= \int_0^1 \left(\prod_{j' \neq j} t^{q_{j'}^{(i)}}\right) q_j^{(i)} t^{q_j^{(i)}-1} dt \\
&= q_j^{(i)} \int_0^1 t^{(n-i+1)-1} dt = \frac{q_j^{(i)}}{n-i+1},
\end{aligned}$$

where we use that $\sum_{j \in M^{(i)}} q_j^{(i)} = n - i + 1$ for any value of i . \blacktriangleleft

Now, the crucial observation is that the expected contribution of any buyer to the social welfare in the posted-prices mechanism is at least as large as under the quantile allocation rule. To see this, fix buyer i and split buyer i 's contribution to the social welfare into revenue and utility. Concerning revenue, note that in both cases the probability of selling any item j to buyer i is equal to $\frac{q_j^{(i)}}{n-i+1}$ and we allocate one item for sure. So, the expected revenue is identical. Further, since we maximize utility in the posted-prices mechanism, the achieved utility is always at least the utility of the quantile allocation rule. So, overall, we get $\mathbf{E}[\text{SW}_{\text{quantile}}] \leq \mathbf{E}[\text{SW}_{\text{pp}}]$.

Now, we aim to control the distribution of $v_{i,j}$ given that $X_{i,j} = 1$ in order to get access to the value of an agent being allocated an item in the quantile allocation rule. To this end, we use the following lemma.

► **Lemma 3.** *For all i, j and $M^{(i)}$, we have*

$$\Pr[v_{i,j} \leq t \mid X_{i,j} = 1, M^{(i)}] = F_j(t)^{\frac{n-i+1}{q_j^{(i)}}}$$

Proof. Observe that in the vector $(R_j^{(i)})_{j \in M^{(i)}}$, we choose j to maximize $R_j^{(i)}$. Now, for any value $v_{i,j'}$, we consider the following transform ψ_j : For any $j' \in M^{(i)}$, define

$$\psi_j(v_{i,j'}) := F_j^{-1} \left((R_{j'}^{(i)})^{q_j^{(i)}} \right).$$

Observe that for $j' = j$, we get that

$$\psi_j(v_{i,j}) = F_j^{-1} \left((R_j^{(i)})^{q_j^{(i)}} \right) = F_j^{-1} \left(\left(F_j(v_{i,j})^{\frac{1}{q_j^{(i)}}} \right)^{q_j^{(i)}} \right) = F_j^{-1} (F_j(v_{i,j})) = v_{i,j}.$$

Further, we can compute the CDF as

$$\begin{aligned}
\Pr[\psi_j(v_{i,j'}) \leq t] &= \Pr \left[F_j^{-1} \left((R_{j'}^{(i)})^{q_j^{(i)}} \right) \leq t \right] = \Pr \left[R_{j'}^{(i)} \leq F_j(t)^{\frac{1}{q_j^{(i)}}} \right] \\
&= \Pr \left[F_{j'}(v_{i,j'}) \leq F_j(t)^{\frac{q_{j'}^{(i)}}{q_j^{(i)}}} \right] = F_j(t)^{\frac{q_{j'}^{(i)}}{q_j^{(i)}}},
\end{aligned}$$

where in the last step, we used that $F_{j'}(v_{i,j'}) \sim \text{Unif}[0, 1]$. As a consequence,

$$\begin{aligned}
 \Pr[v_{i,j} \leq t \mid X_{i,j} = 1, M^{(i)}] &= \Pr[\psi_j(v_{i,j}) \leq t \mid \psi_j(v_{i,j}) > \psi_j(v_{i,j'}) \text{ for } j \neq j', M^{(i)}] \\
 &= \Pr\left[\max_{j' \in M^{(i)}} (\psi_j(v_{i,j'})) \leq t \mid M^{(i)}\right] \\
 &= \prod_{j' \in M^{(i)}} \Pr[\psi_j(v_{i,j'}) \leq t] = \prod_{j' \in M^{(i)}} F_j(t)^{\frac{q_{j'}^{(i)}}{q_j^{(i)}}} \\
 &= F_j(t)^{\frac{\sum_{j' \in M^{(i)}} q_{j'}^{(i)}}{q_j^{(i)}}} = F_j(t)^{\frac{n-i+1}{q_j^{(i)}}}.
 \end{aligned}$$

For integral values of $\frac{n-i+1}{q_j^{(i)}}$ (in particular $q_j^{(i)} = 1$), observe that this is exactly the CDF of the maximum of $\frac{n-i+1}{q_j^{(i)}}$ independent draws from distribution F_j .

For the remainder of the proof sketch, let us restrict to the case that $m = n$. Observe that in this special case, we have that all $q_j^{(i)} = 1$, so the probability in the quantile allocation rule of allocating any item $j \in M^{(i)}$ in Observation 2 simplifies to $\frac{1}{n-i+1}$. Therefore,

$$\mathbf{E}[v_{i,j} X_{i,j}] = \frac{n-i+1}{n} \cdot \frac{1}{n-i+1} \cdot \mathbf{E}[v_{i,j} \mid X_{i,j} = 1] = \frac{1}{n} \mathbf{E}[v_{i,j} \mid X_{i,j} = 1].$$

Observe that this argument looks rather innocent in the special case of $m = n$, but requires a much more careful treatment in the general variant: The probabilities $q_j^{(i)}$ are random variables themselves depending on the set $M^{(i)}$. Hence, the calculation can not directly be extended and a more sophisticated argument needs to be applied. In addition, by the above considerations on the quantile allocation rule via Lemma 3, we have that $\mathbf{E}[v_{i,j} \mid X_{i,j} = 1] = \mathbf{E}[\max_{i' \in [n-i+1]} v_{i',j}]$ in the special case of $m = n$. Therefore, we can now simply apply Lemma 4 (see below) to get

$$\mathbf{E}[v_{i,j} X_{i,j}] = \frac{1}{n} \mathbf{E}[v_{i,j} \mid X_{i,j} = 1] = \frac{1}{n} \mathbf{E}\left[\max_{i' \in [n-i+1]} v_{i',j}\right] \geq \frac{1}{n} \cdot \frac{H_{n-i+1}}{H_n} \cdot \mathbf{E}\left[\max_{i' \in [n]} v_{i',j}\right]$$

Note that we take the maximum over i.i.d. random variables. As a consequence, we can conclude by basic calculations:

$$\begin{aligned}
 \mathbf{E}[\text{SW}_{\text{pp}}] &\geq \mathbf{E}[\text{SW}_{\text{quantile}}] = \sum_{i=1}^n \sum_{j=1}^n \mathbf{E}[v_{i,j} X_{i,j}] \geq \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n \frac{H_{n-i+1}}{H_n} \mathbf{E}\left[\max_{i' \in [n]} v_{i',j}\right] \\
 &= \frac{\sum_{i=1}^n H_{n-i+1}}{n H_n} \sum_{j=1}^n \mathbf{E}\left[\max_{i' \in [n]} v_{i',j}\right] = \left(1 - O\left(\frac{1}{\log n}\right)\right) \sum_{j=1}^n \mathbf{E}\left[\max_{i' \in [n]} v_{i',j}\right] \\
 &\geq \left(1 - O\left(\frac{1}{\log n}\right)\right) \mathbf{E}[\text{SW}_{\text{opt}}]
 \end{aligned}$$

Observe that in the general version, comparing to $\sum_{j=1}^n \mathbf{E}[\max_{i' \in [n]} v_{i',j}]$ is a far too strong benchmark. Therefore, we consider an ex-ante relaxation of the offline optimum. As a new technical tool, we introduce in Lemma 5 (see below) an appropriate bound which allows to lower bound the expected maximum of k draws from an MHR distribution by a suitable fraction of $\mathbf{E}[v_{i,j} \mid v_{i,j} \geq F^{-1}(1-q)]$ for any choice of $q \in [0, 1]$. Applying this, we can lower bound the expected contribution of any item j to the quantile welfare by a suitable fraction of its contribution to the offline optimum.

We conclude by giving the lemmas which were used in the proof. First, let us restate a useful lemma from Babaioff et al. [4]. It allows to compare the expectation of the maximum of n and $n' \leq n$ draws from independent and identically distributed random variables, if the distribution has a monotone hazard rate.

► **Lemma 4** (Lemma 5.3 in [4]). *Consider a collection $(X_i)_i$ of independent and identically distributed random variables with a distribution with monotone hazard rate. Then, for any $n' \leq n$, we have*

$$\frac{\mathbf{E} [\max_{i \in [n']} X_i]}{\mathbf{E} [\max_{i \in [n]} X_i]} \geq \frac{H_{n'}}{H_n} \geq \frac{\log n'}{\log n} .$$

In addition, we make use of the following lemma which is used in order to make a suitable comparison to the ex-ante relaxation.

► **Lemma 5.** *Let $z \in [0, 1]$ and $k \in \mathbb{N}$. Further, let \mathcal{D} be a distribution with monotone hazard rate with CDF F , let $X, (Y_i)_i \sim \mathcal{D}$ be independent and identically distributed. For $\alpha \geq \frac{1 + \ln(\frac{1}{z})}{H_k}$, $\alpha \geq 1$, and $\alpha k \leq \frac{1}{z}$, we have $\mathbf{E} [X \mid X \geq F^{-1}(1 - z)] \leq \alpha \mathbf{E} [\max_{i \in [k]} Y_i]$.*

The proof of this lemma can be found in the full version of the paper.

3.2 Static prices

Next, we would like to demonstrate how to use static prices. We consider the case that the number of items m is upper bounded by $\frac{n}{(\log \log n)^2}$. We set the price for item j to

$$p_j = F_j^{-1}(1 - q) , \text{ where } q = \frac{\log \log n}{n} ,$$

which allows us to state the following theorem.

► **Theorem 6.** *The posted-prices mechanism with static prices and independent item-valuations is $1 - O\left(\frac{\log \log \log n}{\log n}\right)$ -competitive with respect to social welfare.*

As before, we defer the proof of this theorem to the full version of the paper and give a quick sketch here: First, observe that we can bound the probability of selling item j to buyer i by the probability of the event that buyer i has only non-negative utility for this item. This implies a bound on the probability of selling item j in our mechanism. Finally, we combine this with a lower bound on the price p_j and hence are able to bound the revenue (and thus the welfare) obtained by our mechanism. Observe that our guarantee only applies if the number of items m is bounded by $\frac{n}{(\log \log n)^2}$. We leave the extension to the general case as an open problem. As a first step, one could try to derive a suitable bound on the utility of agents in order to extend the result.

4 Asymptotically Tight Bounds for Separable Valuations

Let us now come to *separable valuations*, which are common in ad auctions [14, 27]. That is, in order to determine buyer i 's value for item j , let each buyer i have a type $v_i \geq 0$ and let each item have an item-dependent multiplier α_j which can be interpreted as a click through rate in online advertising. Buyer i 's value $v_{i,j}$ for being assigned item j is given by $\alpha_j \cdot v_i$. Without loss of generality, we assume that $\alpha_1 \geq \alpha_2 \geq \dots$ and that $m = n$. The former can be ensured by reordering the items; the latter by adding items with $\alpha_j = 0$ or removing all

items $j \in M$ with $j > n$ respectively. Observe that in the case of $m > n$, items of index larger than n are not matched in either the optimum, nor is it beneficial to match one of these items and leave an item $j \leq n$ unmatched. Note that this correlated setting also contains the single-item scenario as a special case since it can be modeled by $\alpha_1 = 1, \alpha_2 = \dots = 0$. More generally, k identical items can be modeled by $\alpha_1 = \dots = \alpha_k = 1, \alpha_{k+1} = \dots = 0$.

The types $v_1, \dots, v_n \geq 0$ are non-negative, independent and identically distributed random variables with a continuous distribution satisfying the MHR condition. Let j_i denote the item allocated to buyer i and $j_i = m + 1$ if buyer i is not allocated any item where $\alpha_{m+1} = 0$. We can specify the expected social welfare of the matching computed by the mechanism as $\mathbf{E}[\sum_{i=1}^n \alpha_{j_i} v_i] =: \mathbf{E}[\text{SW}_{\text{pp}}]$.

Additionally, the structure of the optimal matching can be stated explicitly. Given any type profile $v = (v_1, \dots, v_n)$, we let $v_{(k)}$ denote the k -th highest order statistics. That is, $v_{(k)}$ is the largest x such that there are at least k entries in v whose value is at least x . Denote its expectation by $\mathbf{E}[v_{(k)}] = \mu_k$. The allocation that maximizes social welfare assigns item 1 to a buyer of type $v_{(1)}$, item 2 to a buyer of type $v_{(2)}$ and so on. Hence, the expected optimal social welfare is given by $\mathbf{E}[\text{SW}_{\text{opt}}] = \mathbf{E}\left[\sum_{j=1}^m \alpha_j v_{(j)}\right] = \sum_{j=1}^m \alpha_j \mu_j$.

4.1 Dynamic prices

First, we focus on posted-prices mechanisms with dynamic prices. Consider step i and buyer i arrives. Let $M^{(i)}$ be the set of remaining items at this time. Our choice of prices ensures that in each step one item is allocated. Therefore, always $|M^{(i)}| = n - i + 1$.

We choose prices $(p_j^{(i)})_{j \in M^{(i)}}$ with the goal that each item is allocated with probability $\frac{1}{n-i+1}$. To this end, let $M^{(i)} = \{\ell_1, \dots, \ell_{n-i+1}\}$ with $\ell_1 < \ell_2 < \dots < \ell_{n-i+1}$ and set

$$p_j^{(i)} = \sum_{k: j \leq \ell_k \leq n-i} (\alpha_{\ell_k} - \alpha_{\ell_{k+1}}) F^{-1}\left(1 - \frac{k}{n-i+1}\right).$$

Given this pricing scheme, we can state the following theorem.

► **Theorem 7.** *The posted-prices mechanism with dynamic prices and separable valuations is $1 - O\left(\frac{1}{\log n}\right)$ -competitive with respect to social welfare.*

The proof can be found in the full version of the paper. First, observe that in principle, buyers will be indifferent between two items j and j' if $\alpha_j = \alpha_{j'}$. As these items are indistinguishable for later buyers anyway and new prices will be defined, we can assume that ties are broken in our favor. That is why we can assume that buyer i will prefer item ℓ_k if and only if $F^{-1}\left(1 - \frac{k}{n-i+1}\right) \leq v_i < F^{-1}\left(1 - \frac{k-1}{n-i+1}\right)$.

Using a suitable lower bound for $F^{-1}\left(1 - \frac{k}{n-i+1}\right)$ via the MHR property, we get a lower bound for the value $v_{i,j}$ if i is matched to j . To this end, we compare quantiles of MHR distributions to the respective order statistics. As stated before, by Section 5, the competitive ratio is optimal.

4.2 Static prices

When restricting to the case of static prices, we define probabilities q_j having the interpretation that a buyer drawn from the distribution has one of items $1, \dots, j$ as their first choice. For technical reasons, we discard items $\hat{m} + 1, \dots, n$ for $\hat{m} = n - n^{5/6}$ by setting $p_j = \infty$ for

these items. For $j \leq \widehat{m}$, we set prices

$$p_j = \sum_{k=j}^n (\alpha'_k - \alpha'_{k+1}) F^{-1}(1 - q_k) \quad , \quad \text{where } q_k = \min \left\{ \frac{k}{n} 2 \log \log n, \frac{k}{n} + \sqrt{\frac{\log n}{n}} \right\} \quad ,$$

where $\alpha'_k = \alpha_k$ for $k \leq \widehat{m}$ and 0 otherwise.

Note the similarity of this price definition to the payments when applying the VCG mechanism. There, the buyer being assigned item j has to pay $\sum_{k=j}^n (\alpha'_k - \alpha'_{k+1}) v_{(k+1)}$.

► **Theorem 8.** *The posted-prices mechanism with static prices and separable valuations is $1 - O\left(\frac{\log \log \log n}{\log n}\right)$ -competitive with respect to social welfare.*

The proof of this theorem can be found in the full version of the paper. The general steps are as follows. We first show that our prices are fairly low, meaning that the probability of selling all items $1, \dots, \widehat{m}$ is reasonably high. Having this, we decompose the social welfare into utility and revenue. The revenue of our mechanism is bounded in terms of the VCG revenue. To this end, we use that our pricing rule is quantile-based and exploit that the quantiles of any MHR distributions are lower-bounded by suitable fractions of expected order statistics. Talking about utility, we use a link to Myerson's theory and virtual values in order to achieve our desired bound. Again, by Section 5, the competitive ratio is asymptotically tight.

5 Asymptotically Upper Bounds on the Competitive Ratios

Our competitive ratios are asymptotically tight. In this section we provide matching upper bounds showing optimality. To this end, we consider the case of selling a single item with static and dynamic prices respectively. In any of the two cases, we can achieve asymptotic upper bounds on the competitive ratio of posted prices mechanisms which match our results from the previous sections. In particular, we prove that these bounds hold for any choice of pricing strategy.

5.1 Dynamic prices

We consider the guarantee of our dynamic-pricing mechanisms first. Even with a single item and types drawn from an exponential distribution, the best competitive ratio is $1 - \Omega\left(\frac{1}{\log n}\right)$. We simplify notation by omitting indices when possible.

► **Proposition 9.** *Let $v_1, \dots, v_n \in \mathbb{R}_{\geq 0}$ be random variables where each v_i is drawn i.i.d. from the exponential distribution with rate 1, i.e., $v_1, \dots, v_n \sim \text{Exp}(1)$. For all dynamic prices, the competitive ratio of the mechanism picking the first v_i with $v_i \geq p^{(i)}$ is upper bounded by $1 - \Omega\left(\frac{1}{\log n}\right)$.*

In order to prove Proposition 9, we use that the expected value of the optimal offline solution (the best value in hindsight) is given by $\mathbf{E} [\max_{i \in [n]} v_i] = H_n$ [3]. Therefore, it suffices to show that the expected value of any dynamic pricing rule is upper bounded by $H_n - c$ for some constant $c > 0$.

To upper-bound the expected social welfare of any dynamic pricing rule, we use the fact that this problem corresponds to a Markov decision process and the optimal dynamic prices are given by³

$$p^{(n)} = 0 \quad \text{and} \quad p^{(i)} = \mathbf{E} [\max\{v_{i+1}, p^{(i+1)}\}] \quad \text{for } i < n \quad .$$

³ To the best of our knowledge, this is a folklore result.

Furthermore, $p^{(0)}$ is exactly the expected social welfare of this mechanism. Therefore, the following lemma with $k = n$ directly proves our claim.

► **Lemma 10.** *Let $v_1, \dots, v_n \in \mathbb{R}_{\geq 0}$ be random variables where each v_i is drawn i.i.d. from the exponential distribution $\text{Exp}(1)$. Moreover, let $p^{(n)} = 0$ and $p^{(i)} = \mathbf{E} [\max\{v_{i+1}, p^{(i+1)}\}]$ for $i < n$. Then, we have $p^{(n-k)} \leq H_k - \frac{1}{8}$ for all $2 \leq k \leq n$.*

The proof via induction over k can be found in the full version of the paper.

5.2 Static prices

For static pricing rules, we show that any mechanism is $1 - \Omega\left(\frac{\log \log \log n}{\log n}\right)$ -competitive. Again, this bound even holds for a single item and the valuations being drawn from an exponential distribution.

► **Proposition 11.** *Let $v_1, \dots, v_n \in \mathbb{R}_{\geq 0}$ be random variables where each v_i is drawn i.i.d. from the exponential distribution with rate 1, i.e., $v_1, \dots, v_n \sim \text{Exp}(1)$. For all static prices $p \in \mathbb{R}_{\geq 0}$ the competitive ratio of the mechanism picking the first v_i with $v_i \geq p$ is upper bounded by $1 - \Omega\left(\frac{\log \log \log n}{\log n}\right)$.*

The proof of Proposition 11 can be found in the full version of the paper. The idea is as follows. The expected welfare obtained by the static-price mechanism using price p is given by $\mathbf{E} [\text{SW}_{\text{pp}}] = \mathbf{E} [v \mid v \geq p] \cdot \Pr [\exists i : v_i \geq p] = (p + 1) \cdot (1 - (1 - e^{-p})^n)$. This has to be compared to the expected value of the optimal offline solution (the best value in hindsight), which is given by $\mathbf{E} [\max_{i \in [n]} v_i] = H_n$ [3].

6 Extensions to Subadditive Buyers and Revenue Considerations

In this section, we illustrate that the same style of mechanisms used for unit-demand buyers in principle is also applicable for subadditive buyers. A valuation function v_i is subadditive if $v_i(S \cup T) \leq v_i(S) + v_i(T)$ for any $S, T \subseteq M$. This generalizes unit-demand functions considered so far in this paper. Instead of being interested in only a single-item, each buyer now has a subadditive valuation function over item bundles and can thus be interested in multiple items.

To generalize the MHR property, we assume that the subadditive valuation functions are drawn from distributions with MHR marginals. That is, $v_i \sim \mathcal{D}$ and we assume that $v_i(\{j\})$ has a marginal distribution with monotone hazard rate. Buyers arrive online and purchase the bundle of items which maximizes the buyer's utility.

We can construct a dynamic-pricing mechanism which is $1 - O\left(\frac{1 + \log m}{\log n}\right)$ -competitive. For a detailed explanation, we refer to the full version of the paper. The general approach is to split the set of buyers in subgroups of size $\lfloor \frac{n}{m} \rfloor$ and sell each item to one of these groups. For the k -th buyer in every group, the price for the item in question is set to $p_j^{(k)} = F_j^{-1}\left(1 - \frac{1}{\lfloor \frac{n}{m} \rfloor - k + 1}\right)$, where F_j^{-1} denotes the quantile function of the marginal distribution of $v_i(\{j\})$. Using techniques similar to the ones in the unit-demand case allows to bound the revenue of the posted-prices mechanism by the desired fraction of the optimal social welfare. Hence, the argument directly implies the respective bounds for welfare and revenue.

In the static pricing environment, our results can be extended to a mechanism which is $1 - O\left(\frac{\log \log \log n}{\log n} + \frac{\log m}{\log n}\right)$ -competitive for subadditive buyers. Details can be found in the full version. As before, let F_j^{-1} be the quantile function of the marginal distribution of

$v_i(\{j\})$. Setting fairly low prices of $p_j = F_j^{-1}(1 - q)$ for $q = \frac{m \log \log n}{n}$ ensures that we sell all items with a suitably high probability. Afterwards, we can apply the same bounds for MHR distributions as in the previous sections in order to bound the revenue of the mechanism with the respective fraction of the optimal social welfare. Again, this directly implies the mentioned competitive ratio for welfare as well as revenue, as the revenue of any individually rational mechanism is upper-bounded by the corresponding social welfare.

Note that the guarantees now depend on the number of items m . To make them meaningful, we need $m = o(n)$. This makes them significantly worse than the ones we obtain for unit-demand functions with a much more careful treatment. However, they are stronger in one aspect, namely that in both cases we bound the revenue of the mechanism in terms of the optimal social welfare. In particular, this means that they are also approximations of the optimal revenue. Interestingly, the optimality results from Section 5 also transfer.

References

- 1 Melika Abolhassani, Soheil Ehsani, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Robert D. Kleinberg, and Brendan Lucier. Beating $1-1/e$ for ordered prophets. 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 61–71. ACM, 2017. doi:10.1145/3055399.3055479.
- 2 Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM J. Comput.*, 43(2):930–972, 2014. doi:10.1137/120878422.
- 3 Barry C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics (Classics in Applied Mathematics)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- 4 Moshe Babaioff, Liad Blumrosen, Shaddin Dughmi, and Yaron Singer. Posting prices with unknown distributions. *ACM Trans. Econ. Comput.*, 5(2), 2017. doi:10.1145/3037382.
- 5 Alexander Braun, Matthias Buttkus, and Thomas Kesselheim. Asymptotically optimal welfare of posted pricing for multiple items with mhr distributions, 2021. arXiv:2107.00526.
- 6 Yang Cai and Mingfei Zhao. Simple mechanisms for subadditive buyers via duality. 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 170–183. ACM, 2017. doi:10.1145/3055399.3055465.
- 7 Shuchi Chawla, Nikhil R. Devanur, Alexander E. Holroyd, Anna R. Karlin, James B. Martin, and Balasubramanian Sivan. Stability of service under time-of-use pricing. 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 184–197. ACM, 2017. doi:10.1145/3055399.3055455.
- 8 Shuchi Chawla, Jason D. Hartline, and Robert D. Kleinberg. Algorithmic pricing via virtual valuations. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 243–251. ACM, 2007. doi:10.1145/1250910.1250946.
- 9 Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. 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 311–320. ACM, 2010. doi:10.1145/1806689.1806733.
- 10 Edward Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, 1971. URL: <https://EconPapers.repec.org/RePEc:kap:pubcho:v:11:y:1971:i:1:p:17-33>.
- 11 José R. Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Posted price mechanisms for a random stream of customers. In Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin, editors, *Proceedings of the 2017 ACM Conference on*

- Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 169–186. ACM, 2017. doi:10.1145/3033274.3085137.
- 12 José R. Correa, Patricio Foncea, Dana Pizarro, and Victor Verdugo. From pricing to prophets, and back! *Oper. Res. Lett.*, 47(1):25–29, 2019. doi:10.1016/j.orl.2018.11.010.
 - 13 Paul Dütting, Thomas Kesselheim, and Brendan Lucier. An $o(\log \log m)$ prophet inequality for subadditive combinatorial auctions. *SIGecom Exch.*, 18(2):32–37, 2020. doi:10.1145/3440968.3440972.
 - 14 B. Edelman, M. Ostrovsky, and M. Schwarz. Selling billions of dollars worth of keywords: The generalized second-price auction. *American Economic Review*, 97(1):242–259, 2007.
 - 15 Soheil Ehsani, MohammadTaghi Hajiaghayi, Thomas Kesselheim, and Sahil Singla. Prophet secretary for combinatorial auctions and matroids. 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 700–714. SIAM, 2018. doi:10.1137/1.9781611975031.46.
 - 16 Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. 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 123–135. SIAM, 2015. doi:10.1137/1.9781611973730.10.
 - 17 Yiannis Giannakopoulos and Keyu Zhu. Optimal pricing for MHR distributions. In George Christodoulou and Tobias Harks, editors, *Web and Internet Economics - 14th International Conference, WINE 2018, Oxford, UK, December 15-17, 2018, Proceedings*, volume 11316 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2018. doi:10.1007/978-3-030-04612-5_11.
 - 18 Theodore Groves. Incentives in teams. *Econometrica*, 41(4):617–31, 1973. URL: <https://EconPapers.repec.org/RePEc:ecm:emetrp:v:41:y:1973:i:4:p:617-31>.
 - 19 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 58–65. AAAI Press, 2007. URL: <http://www.aaai.org/Library/AAAI/2007/aaai07-009.php>.
 - 20 Theodore P Hill, Robert P Kertz, et al. Comparisons of stop rule and supremum expectations of iid random variables. *The Annals of Probability*, 10(2):336–345, 1982.
 - 21 Yaonan Jin, Weian Li, and Qi Qi. On the approximability of simple mechanisms for mhr distributions. In Ioannis Caragiannis, Vahab Mirrokni, and Evdokia Nikolova, editors, *Web and Internet Economics*, pages 228–240, Cham, 2019. Springer International Publishing.
 - 22 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bull. Amer. Math. Soc.*, 83(4), 1977.
 - 23 Brendan Lucier. An economic view of prophet inequalities. *SIGecom Exchanges*, 16(1):24–47, 2017. doi:10.1145/3144722.3144725.
 - 24 Roger B. Myerson. Optimal auction design. *Math. Oper. Res.*, 6(1):58–73, 1981. doi:10.1287/moor.6.1.58.
 - 25 Horst Rinne. *The Hazard rate : Theory and inference (with supplementary MATLAB-Programs)*. Justus-Liebig-Universität, 2014. URL: <http://geb.uni-giessen.de/geb/volltexte/2014/10793>.
 - 26 E. Samuel-Cahn. Comparison of threshold stop rules and maximum for independent nonnegative random variables. *Annals of Probability*, 12:1213–1216, 1984.
 - 27 H. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, 2007.
 - 28 William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961. URL: <https://EconPapers.repec.org/RePEc:bla:jfinan:v:16:y:1961:i:1:p:8-37>.

- 29 Hanrui Zhang. Improved Prophet Inequalities for Combinatorial Welfare Maximization with (Approximately) Subadditive Agents. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.82.

Covert Computation in Staged Self-Assembly: Verification Is PSPACE-Complete

David Caballero ✉

Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Timothy Gomez ✉

Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Robert Schweller ✉

Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Tim Wylie ✉

Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Abstract

Staged self-assembly has proven to be a powerful abstract model of self-assembly by modeling laboratory techniques where several nanoscale systems are allowed to assemble separately and then be mixed at a later stage. A fundamental problem in self-assembly is Unique Assembly Verification (UAV), which asks whether a single final assembly is uniquely constructed. This has previously been shown to be Π_2^P -hard in staged self-assembly with a constant number of stages, but a more precise complexity classification was left open related to the polynomial hierarchy.

Covert Computation was recently introduced as a way to compute a function while hiding the input to that function for self-assembly systems. These Tile Assembly Computers (TACs), in a growth only negative aTAM system, can compute arbitrary circuits, which proves UAV is coNP-hard in that model. Here, we show that the staged assembly model is capable of covert computation using only 3 stages. We then utilize this construction to show UAV with only 3 stages is Π_2^P -hard. We then extend this technique to open problems and prove that general staged UAV is PSPACE-complete. Measuring the complexity of n stage UAV, we show Π_{n-1}^P -hardness. We finish by showing a Π_{n+1}^P algorithm to solve n stage UAV leaving only a constant gap between membership and hardness.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Problems, reductions and completeness

Keywords and phrases self-assembly, covert computation, staged self-assembly, assembly verification

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.23

Funding This research was supported in part by National Science Foundation Grant CCF-1817602.

1 Introduction

The *Staged Self-Assembly* model was designed as an extension to the standard hierarchical model of tile self assembly that mimics the abilities of scientists in the lab to control the assembly process by mixing test tubes. The additional features in this model allow for more efficient tile complexity, but increased complexity of certain verification problems.

We use the concept of *Covert Computation*, a requirement of a computational system stipulating that the input and computational history of the computation be hidden in the final output of the system, within the context of Staged Self-assembly, an extension to tile self-assembly that allows for basic operations such as mixing self-assembly batches over a sequence of distinct stages. We use this connection to resolve open questions regarding the complexity of the *Unique Assembly Verification* (UAV) problem within staged self-assembly—the problem of whether a given system uniquely produces a specific assembly. The importance of this work stems from the fundamental nature of the UAV problem, along with the natural and experimentally motivated Staged Self-Assembly model. Further, the novel approach by



© David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 23; pp. 23:1–23:18

Leibniz International Proceedings in Informatics



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

which our results are obtained, by way of designing Covert Computation systems in Staged Self-Assembly, may be of independent interest as it shows how to utilize Staged Self-Assembly to implement general purpose computing systems with strong guarantees that might be useful for cryptography or have applications for privacy within biomedical computation.

Staged Self-Assembly. The Staged Self-Assembly model [1, 6, 7, 8, 9, 10, 11, 14, 18] is a generalization of the (2-handed) tile assembly model [4] where particles are modeled by 4-sided Wang tiles which nondeterministically combine based on the affinity of tile edges. Tile self-assembly is a well-studied mathematical abstraction used in the study of self-assembly systems with algorithmically complex behavior, and enjoys experimental success through a DNA implementation [19]. In order to add the basic functionality of what an experimentalist with a set of test tubes could execute [17], the staged model extends tile self-assembly by allowing assembly to occur in multiple separate *bins*, and for the contents of these bins to be either combined or split into a new set of bins after each one of a given sequence of *stages*.

Covert Computation. Tile self-assembly can be used as a model of computation in which tiles attach to an input *seed* structure to grow a final output structure encoding the result of the computation. This basic paradigm is one of most promising avenues for the development of nanoscale molecular computing systems (see [19] for recent experimental work using DNA tiles to implement 6-bit circuits). The authors in [5] recently proposed a new constraint on such computing systems termed *Covert Computation*. A covert computation system computes a function with the additional constraint that the output assembly provides no information about either the original input or the computational history, beyond the actual output of the computed function. This is a particularly daunting self-assembly problem since the output is provided in the form of a self-assembled structure that encodes the exact geometric location of every placed tile. In previous methods of tile self-assembly computation, the entire computational history and original input are easily interpreted from the final output assembly. However, while the output assembly specifies the location of each placed tile, the result of the computation can be a function of not just these tile *locations*, but also of the *order* in which these tiles are placed, which is the technique exploited in [5]. This concept provides a useful technique for proving complexity results, and we use it here to show PSPACE-completeness of verifying unique assembly in staged self-assembly.

Unique Assembly Verification. One well-studied problem in tile self-assembly is the Unique Assembly Verification (UAV) problem which asks if a given system uniquely produces a given assembly. This problem was shown to be solvable in polynomial time in the Abstract Tile Assembly Model [2]. The addition of negative interactions and detachment of tiles makes the UAV problem undecidable [12], while *growth-only* systems with no detachments are coNP-complete [5]. The UAV problem in the 2-Handed Assembly Model was first studied in [4] where coNP membership was shown with coNP-completeness in the third dimension. The problem was also shown to be coNP-complete with a variable temperature [15], but constant temperature UAV in the 2HAM is still open. In the staged assembly model, initial investigation in [16] showed coNP-hardness using four stages and Π_2^P -hardness for seven stages. They also showed membership in PSPACE with a conjecture of PSPACE-completeness.

Our Results. In this paper, we introduce the concept of covert computation in the context of staged self-assembly for the purpose of establishing the complexity of unique assembly verification within the model. First, we show that staged self-assembly is capable of covert

■ **Table 1** Complexities of Unique Assembly Verification in the Staged Assembly Model with respect to the number of stages n . Our results are in bold. *This result uses the temperature as an input parameter/variable for the problem. All other results are true even with a constant temperature.

Stages	Membership		Hardness	
1 (2HAM)	coNP	In [4]	coNP-complete*	In [15]
2	Π_3^P	Thm. 19	coNP-hard*	In [15]
3	Π_4^P	Thm. 19	Π_2^P -hard	Thm. 6
$n > 3$	Π_{n+1}^P	Thm. 19	Π_{n-1}^P -hard	Thm. 12
General	PSPACE	In. [16]	PSPACE-complete	Thm. 10

computation even when limited to three stages. Next, we use this fact to show UAV is PSPACE-complete in staged self-assembly, resolving the open problem from [16]. Along the way, we improve on some results from [16]: we show that UAV is Π_2^P -hard with just three stages, improving on the previous hardness result requiring seven stages. We then generalize this result to show that for n stages, UAV is Π_{n-1}^P -hard, but yields a Π_{n+1}^P algorithm, leaving only a gap of two in levels between membership and hardness for this problem. An overview of our results and known results related to UAV is shown in Table 1.

2 Preliminaries

We first provide definitions for the staged self-assembly model and covert computation.

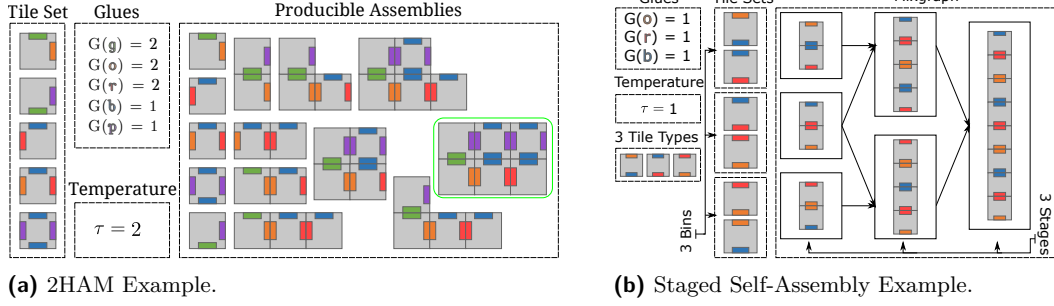
Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ . Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations, bond graphs, and stability. A *configuration* is a partial function $A : \mathbb{Z}^2 \rightarrow T$ for some set of tiles T , i.e. an arrangement of tiles on a square grid. For a given configuration A , define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is said to be τ -stable for positive integer τ if every edge cut of G_A has strength at least τ , and is τ -unstable otherwise.

Assemblies. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $A \circ f$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations A and B , B is a *translation* of A , written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector \vec{u} . For a configuration A , the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly \tilde{A} is a *subassembly* of an assembly \tilde{B} , denoted $\tilde{A} \subseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. An assembly is τ -stable provided the configurations it contains are τ -stable. Assemblies \tilde{A} and \tilde{B} are τ -combinable into an assembly \tilde{C} provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$, $A \cap B = \emptyset$, and \tilde{C} is τ -stable.

Two-handed assembly and bins. We define the assembly process in terms of bins. A *bin* is an ordered tuple (S, τ) where S is a set of *initial* assemblies and τ is a positive integer parameter called the *temperature*. For a bin (S, τ) , the set of *produced* assemblies $P'_{(S, \tau)}$ is defined recursively as follows:

1. $S \subseteq P'_{(S, \tau)}$.
2. If $A, B \in P'_{(S, \tau)}$ are τ -combinable into C , then $C \in P'_{(S, \tau)}$.



■ **Figure 1** (a) A 2HAM example that uniquely builds a 2×3 rectangle. The top 4 tiles in the tile set all combine with strength-2 glues building the ‘L’ shape. The tile with blue and purple glues needs two tiles to cooperatively bind to the assembly with strength 2. All possible producibles are shown with the terminal assembly highlighted. (b) A simple staged self-assembly example. The system has 3 bins and 3 stages, as shown in the mixgraph. There are three tile types in our system that we assign to bins as desired. From each stage only the terminal assemblies are added to the next stage. The result of this system is the assembly shown in the bin in stage 3.

A produced assembly is *terminal* provided it is not τ -combinable with any other producible assembly, and the set of all terminal assemblies of a bin (S, τ) is denoted $P_{(S, \tau)}$. Intuitively, $P'_{(S, \tau)}$ represents the set of all possible assemblies that can self-assemble from the initial set S , whereas $P_{(S, \tau)}$ represents only the set of supertiles that cannot grow any further. The assemblies in $P_{(S, \tau)}$ are *uniquely produced* iff for each $x \in P'_{(S, \tau)}$ there exists a corresponding $y \in P_{(S, \tau)}$ such that $x \sqsubseteq y$. Thus unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in $P_{(S, \tau)}$.

Staged assembly systems. An r -stage b -bin mix graph $M_{r, b}$ is an acyclic r -partite digraph consisting of rb vertices $m_{i, j}$ for $1 \leq i \leq r$ and $1 \leq j \leq b$, and edges of the form $(m_{i, j}, m_{i+1, j'})$ for some i, j, j' . A *staged assembly system* is a 3-tuple $\langle M_{r, b}, \{T_1, T_2, \dots, T_b\}, \tau \rangle$ where $M_{r, b}$ is an r -stage b -bin mix graph, T_i is a set of tile types, and τ is an integer temperature parameter.

Given a staged assembly system, for each $1 \leq i \leq r$, $1 \leq j \leq b$, we define a corresponding bin $(R_{i, j}, \tau)$ where $R_{i, j}$ is defined as follows:

1. $R_{1, j} = T_j$ (this is a bin in the first stage);
2. For $i \geq 2$, $R_{i, j} = \left(\bigcup_{k: (m_{i-1, k}, m_{i, j}) \in M_{r, b}} P_{(R_{i-1, k}, \tau)} \right)$.

Thus, the j^{th} bin in stage 1 is provided with the initial tile set T_j , and each bin in any subsequent stage receives an initial set of assemblies consisting of the terminally produced assemblies from a subset of the bins in the previous stage as indicated by the edges of the mix graph.¹ The *output* of the staged system is the union of all terminal assemblies from each of the bins in the final stage.² We say this set of output assemblies is *uniquely produced* if each bin in the staged system uniquely produces its respective set of terminal assemblies.

¹ The original staged model [9] only considered $\mathcal{O}(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage. Since our systems may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.

² This is a slight modification of the original staged model [9] in that the final stage may have multiple bins. However, all of our results apply to both variants of the model.

Covert Computation. Tile assembly computers were first defined in [5, 13] with respect to the aTAM. We provide formal definitions of both Tile Assembly Computers and Covert Computation with respect to the Staged Self-Assembly model.

A Staged Tile Assembly Computer (STAC) for a function f consists of a staged self-assembly system, and a format for encoding the input into tiles sets and a format for reading the output from the terminal assembly. The input format is a specification for what set of tiles to add to a specific bin in the first stage. Each bit of the input must be mapped to one of two sets of tiles for the respective bit position: a tile set representing “0”, or tile set representing “1”. The input set for the entire string is the union of all these tile sets. Our staged self assembly system, with the set of tiles needed to build the input seed added in a designated bin, is our final system which performs the computation. The output of the computation is the terminal assembly the system assembles. To interpret what bit-string is represented by the assembly, a second *output* format specifies a pair of sub-assemblies and locations for each bit. An assembly that represents a bitstring is created by the union of each sub-assembly represented by each bit.

For a STAC to *covertly* compute f , the STAC must compute f and produce a unique assembly for each possible output of f . Thus, for all x such that $f(x) = y$, a covert STAC that computes f produces the same output assembly representing output y for each possible input x , making it impossible to determine which input value x was provided to the system.

Input Template. An n -bit input template over tile set T is a sequence of ordered pairs of tile sets over T : $I = (I_{0,0}, I_{0,1}), \dots, (I_{n-1,0}, I_{n-1,1})$. For a given n -bit string $b = b_0, \dots, b_{n-1}$ and n -bit input template I , the input tile set for b with respect to I is the set $I(b) = \bigcup_i I_{i,b_i}$.

Output Template. An n -bit output template over tile set T is a sequence of ordered pairs of configurations over T : $O = (C_{0,0}, C_{0,1}), \dots, (C_{n-1,0}, C_{n-1,1})$. For a given n -bit string $x = x_0, \dots, x_{n-1}$ and n -bit output template O , the *representation* of x with respect to O is $O(x) =$ the assembly of $\bigcup_i C_{i,x_i}$. A template is valid for a temperature parameter $\tau \in \mathbb{Z}^+$ if this union never contains overlaps for any choice of x , and is always τ -stable. An assembly $B \supseteq O(x)$, which contains $O(x)$ as a subassembly, is said to represent x as long as $O(d) \not\subseteq B$ for any $d \neq x$.

Function Computing Problem. A *staged tile assembly computer* (STAC) is an ordered triple $\mathfrak{S} = (\Gamma, I, O)$ where $\Gamma = (M, \{\emptyset, T_2, \dots, T_i\}, \tau)$ is a staged self assembly system, I is an n -bit input template, and O is a k -bit output template. A STAC is said to compute function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^k$ if for any $x \in \mathbb{Z}_2^n$ and $y \in \mathbb{Z}_2^k$ such that $f(x) = y$, then the staged self assembly system $\Gamma_{\mathfrak{S},x} = (M, \{I(x), T_2, \dots, T_i\}, \tau)$ uniquely assembles a set of assemblies which all represent y with respect to template O .

Covert Computation. A STAC *covertly* computes a function $f(x) = y$ if 1) it computes f , and 2) for each y , there exists a unique assembly A_y such that for all x , where $f(x) = y$, the system $\Gamma_{\mathfrak{S},x} = (M, \{I(x), T_1, \dots, T_i\}, \tau)$ uniquely produces A_y . In other words, A_y is determined by y , and every x where $f(x) = y$ has the exact same final assembly.

3 Covert Computation in Staged Self-assembly

Here, we demonstrate covert computation in the staged assembly model. This construction creates a logic circuit using a 3-stage temperature-2 system with a number of bins polynomial in the size of the circuit. We consider only circuits made up of functionally universal NAND gates, but these techniques could be used to create any 2-input gate.

Figure 2b shows a basic overview of the mixgraph used for the covert computation implementation. The method requires three stages with a polynomial number of mixing bins.

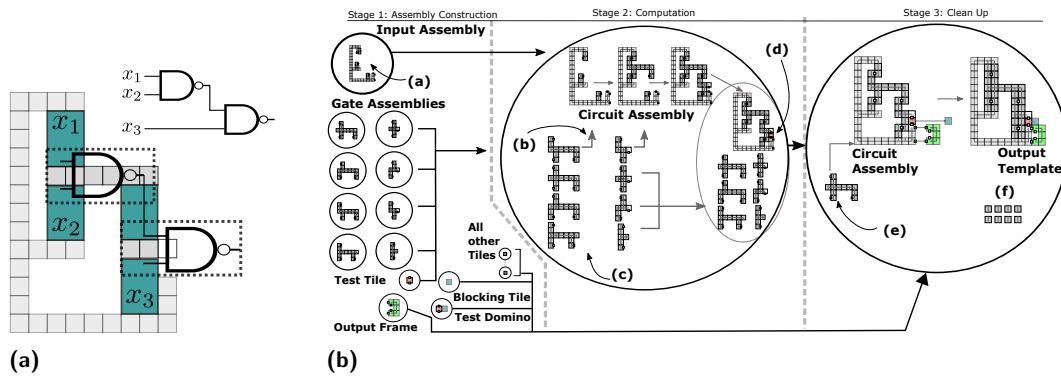
- In the first stage, we assemble the components needed to perform the computation. These include an *Input Assembly*, which encodes the input to the function, *Gate Assemblies*, which act as individual gates and perform the computation via their attachment rules and geometry, and additional assemblies which are used to help “clean up” our circuit and covertly get the output.
- In stage two, the input assembly and gate assemblies are added to a single bin along with a test tile. The gate assemblies will begin to attach to the input assembly creating a *Circuit Assembly*. Once the computation is complete, the test tile can attach to the circuit assembly if and only if the output is true. The circuit assembly is terminal in this bin and will be passed to the final stage.
- The final stage adds additional assemblies to the bin along with most of the tile set as single tiles (not shown in figure). The additional assemblies read the output of the circuit and it grows into one of the output templates. The *Output Frame* searches for the test tile representing the output of the circuit. The single tiles fill in any spaces left in the circuit assembly that would show the computation history, thereby turning the assembly into the output template. This requires a linear number of additional bins in the first and second stage to store these single tiles while mixing takes place in other bins.

For our circuit assembly we implement Planar Logic Circuits with only NAND gates. An example circuit and an assembly showing how the gates are laid out are shown in Figure 2a. Wires are represented by 2×3 blocks of tiles shown in blue in the image. Input and Gate assemblies contain a subset of the tiles in each block we call *arms* which represent the values being passed along the wires. The input assembly is a comb-like structure that is designed so that each input bit reaches the gate it is used at (Figure 3a). For each NAND gate in the circuit we have 4 different assemblies, one for each possible input to the gate. A gate assembly can cooperatively bind to the input assembly if the variable values match. The gate assembly has a third arm that represents the output. This allows the next gate assembly to attach, which continues propagating until the computation is done and the circuit assembly is complete. We now cover the construction in detail by stage.

3.1 First Stage - Assembly Construction

Each bin in the first stage will individually create the assemblies that will come together in the next stage. For an n -input k -gate NAND logic circuit (considering crossovers as three XOR gates [5]), we have an input assembly, $4k$ gate assemblies, and a constant number of other assemblies that will be used in the final stage. Here we will describe the details of the individual assemblies created in addition to the *arms*, which function as wires in our system.

Input. For each bit of the input we have two possible input bit assemblies (Figure 3a). The value of the bit determines which tiles will be added to create that input bit assembly in the first stage. Figure 3a shows the selected assemblies that come together to form the *input assembly* shown in Figure 3b. Each subassembly has a domino which we call an “arm” representing the corresponding bit value. The shape of these assemblies depends on the gates to which they input because the arm of the assembly must reach the location of the gate it inputs to. The last input bit assembly also contains an extra set of tiles that reach the final output gate with a strength-1 glue on its north end and two glues on its east to allow for the test tile and output frame to attach.



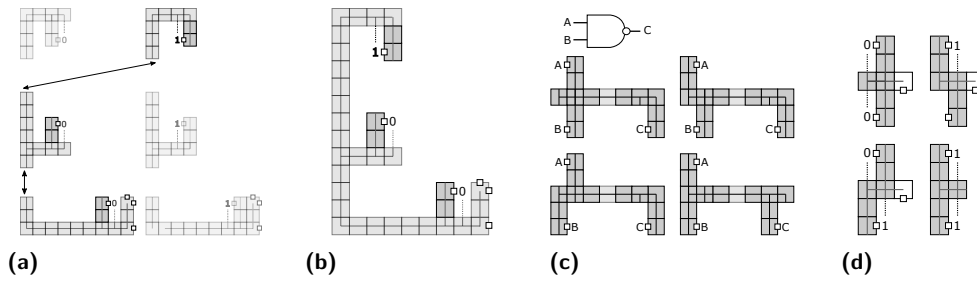
■ **Figure 2** (a) Simple 3-input logic circuit using 2 NAND gates, and the high-level abstraction of the circuit assembly showing the input variables and gates highlighted as blocks. Blue blocks are the sections of the assemblies we call *Arms* that function as wires in the systems. (b) (1) Our input assembly and gate assemblies are constructed in separate bins. (2) Gate assemblies attach to the input assembly forming a circuit assembly. (3) Unused gates are terminal in the second stage. (4) This circuit evaluates to true, so the test tile will be able to attach. (5) Gate assemblies in this stage grow into a circuit using single tiles. (6) Single tiles fill in open spots in the circuit assembly to hide the history. The additional assemblies are used to reach the output template.

Arms. We describe assemblies as having input or output *arms* which function as the wires of our circuit. Arms are vertical dominoes that represent bit values, with their location on the assembly representing the bit having a value 0 or 1 (Figure 4a). The output arm being in the left position represents a bit value of 0, with the right position representing 1. The locations of input arms are complementary (right represents 0, left represents 1) to the output arms. These arms have a glue on the second tile on the inner side. An input arm will attach to an output arm to “read” the bit (Figure 4b) if they represent the same wire and the same value. This glue is a strength-1 glue, so the assembly must attach cooperatively elsewhere in the assembly. Another key feature of these arms is the ability to hide the information passed through by adding single tiles in a later stage. The spaces left by the attachment may be filled by single tiles which results in an assembly which looks like Figure 4c where the value passed cannot be read. This feature will be used in the final stage of our system.

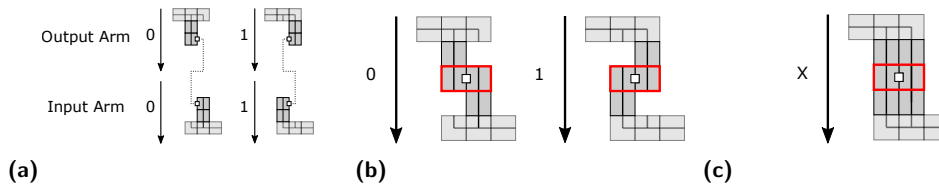
Gates. For each gate we create four assemblies with each representing one of the valid input/output combinations of the desired logic gate. Each gate assembly has two input arms and one output arm. We encode the logic gate by placing the output arm in the column representing the output of the gate when input with the bits represented by the input arms (Figure 3c). This assembly has strength-1 glues on each of its arms. The shape of each gate is dependent on the layout of the circuit since the output arm needs to reach to the next gate. In the case a gate has a fan out (outputs to multiple gates) a gate assembly may have multiple output arms which share arm position. We will refer to the final gate of the circuit as the *output gate*. It does not contain an output arm but instead contains a flag tile to represent an output of false, or no flag tile to represent an output of true which can be seen in Figure 3d. The flag tile also contains a strength-1 glue on it’s south edge which allows for the test tile to attach.

3.2 Second Stage - Computation

In the second stage there is a single bin where the circuit assembly is created. In this stage the input assembly and the gate assemblies are mixed together to compute the encoded circuit. The computation starts by attaching gates to the input assembly to begin to build



■ **Figure 3** (a) Possible input bit assemblies for a 3 bit function. Solid lines between tiles indicate a strength 2 glue between the tiles. Small boxes indicate a strength 1 glue. For each bit we select either the left or the right assembly based on the value of the bit and add those tiles to our input bin in the *First Stage*. Lighter tiles are not used. (b) The input assembly that is constructed in the *First Stage*. The last input bit assembly contains an extra column of tiles that reaches to where the output gate will be for cooperative attachment of the test tile. (c) 4 gate assemblies, one for each possible input combination of a NAND gate. Glues are labeled to match the wires of the NAND gate. (d) Output gates. True output gates contain a flag tile (white).

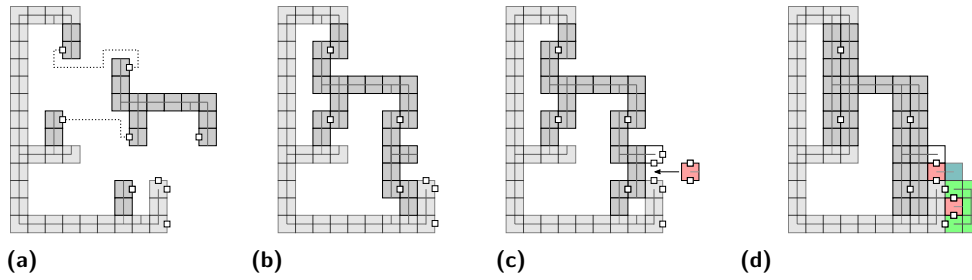


■ **Figure 4** (a) Information being passed along a wire is represented by the position of a domino called an arm. Output arms represent a signal of “1” or “0” by being in the left or right position, respectively. Input arms read bit values and have complimentary arm placement to allow for attachment. (b) Information is passed by attachment. Another assembly may attach if the arms have matching glues (they represent the same wire) and they have complementary arms (represent the same bit value). (c) In the final stage we add additional tiles to hide the information that was passed along a wire.

the circuit assembly. Once both inputs to a gate are present on the circuit assembly, the next gate assembly can cooperatively attach to the circuit assembly since each arm has an attachment strength of 1 as seen in Figure 5a. In this stage we also add the test tile. If the output of the circuit is true, the flag tile can attach as in Figure 5c. If the output is false, the terminal circuit assembly can be seen in Figure 5b. The test assembly is not able to attach to the circuit assembly in this case and will be terminal. We note that at this point it is possible to read the output of the circuit by checking the terminal assemblies, however the computation history can still be read so the covert computation is not complete and we need an additional stage.

3.3 Third Stage - Clean Up

In the third stage we hide the computational history and get the output of the computation. The output template (Full Circuit Assembly) is shown in Figure 5d, which is a circuit assembly with all open spaces in its arm filled in (computation history is hidden) and the additional assemblies attached. The additional assemblies are the Output Frame, which the test tile may attach to, the test domino, which attaches to a circuit assembly but not to the output frame, the blocking tile, which turns a test tile into a test domino, and all single tiles used in the circuit assembly other than the test tile. The difference between the true and false output templates is the inclusion/exclusion of the test tile within the Output Frame.



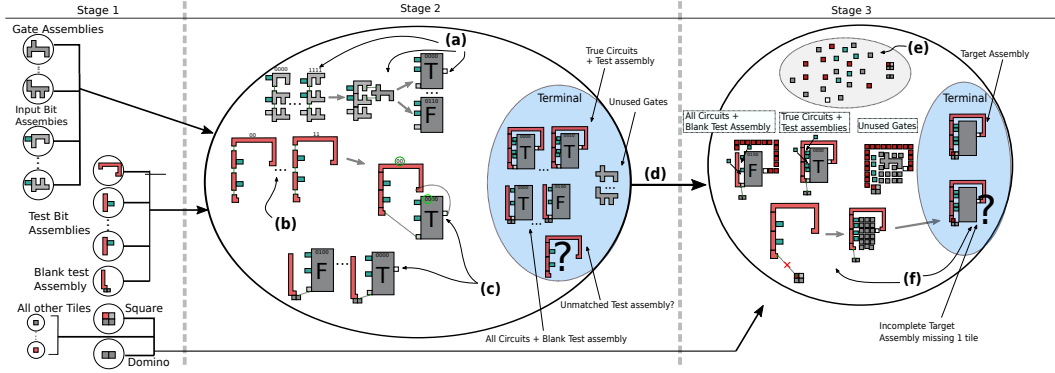
■ **Figure 5** (a) A NAND gate assembly representing input: “10” and output: “1” attaching to the input assembly in the *Second Stage*. (b) False Output Gates which do not contain a flag tile can attach to the circuit if the output is false. This assembly is terminal in the second bin. (c) True Output Gates have an additional flag tile (white) that allows for the test tile (red) to attach cooperatively to the input assembly and the True Output Gate. (d) Single tiles fill in the spaces left by the arms and the output frame attaches forming our target assembly. If there is a flag tile in the output frame the output of this circuit is true. Otherwise, the output is false.

► **Theorem 1.** *For any function f computed by an n -input boolean circuit with k gates, there exists a 3-stage $\mathcal{O}(n^2 + k^2)$ bin, temperature-2 staged tile assembly computer that covertly computes f with an output template size of $\mathcal{O}(n^2 + k^2)$.*

Proof. Given any boolean circuit c , we create gate assemblies for each gate. Given the input to this circuit we create input assemblies that encode the input. In the second stage input assemblies start attaching together to form a circuit assembly. Once two inputs to a gate have attached the gate assembly computing the output of that gate is able to attach. Two gate assemblies cannot attach away from the circuit assembly. There only exist strength-1 glues on the outer edges of a gate assembly, thus a gate can only attach to another assembly with a cooperative bind at each arm. This ensures gates only attach once both inputs are present, and forces the circuit to assemble in the correct order. In the second stage we add in the test tile. This test tile may only attach to a circuit assembly that has a flag tile attached. The flag tile is only present on output gates that evaluate to true so the test tile may attach if and only if the circuit evaluates to true. The test tile is terminal otherwise.

In the final bin we add in single tiles to hide the input to the circuit and the inputs to each gate. As explained above each assembly input to this bin will grow into a full circuit assembly. This full circuit assembly will grow into one of our two output frames. The output frame is a full circuit assembly with the output frame attached. The output frame contains a test tile for false and is empty for true. The terminal assembly of our staged system will have a test tile in the output frame if and only if the circuit evaluated to false which is one of our output frames. If the circuit evaluates to true the test tile will not be present.

This system uses a polynomial number of bins in the first and second stage and a single bin the final stage. The number of bins in the first and second stage are bounded by the size of our tile set since we need individual bins to store each tile so they do not combine before the final stage. The max size of a gate assembly is $\mathcal{O}(\max(n, k))$ since in the worst case a gate needs to stretch across the whole circuit. The same bound applies to input bit assemblies. Therefore the size of the system (the number of tile types + the size of the mix graph) is $\mathcal{O}(n^2 + k^2)$ ◀



■ **Figure 6** A high level overview of the staged system created from an instance of $\forall\text{ESAT}$. (a) An input assembly is created for every possible input to ϕ and is evaluated using the computation technique from Section 3. (b) A test assembly is created for every possible input to X . (c) Test assemblies can attach to a true circuit assembly with the same assignment to X . Blank test assemblies attach to any circuit. (d) Terminal assemblies are passed to the next stage, including unmatched test assemblies if any exist. (e) In this stage we add the domino and square assemblies, as well as every other single tile of the target assembly. (f) Any unmatched test assembly will grow into an incomplete target assembly since it cannot attach to the square assembly. These incomplete target assemblies are terminal, meaning the UAV instance is false.

4 Unique Assembly Verification

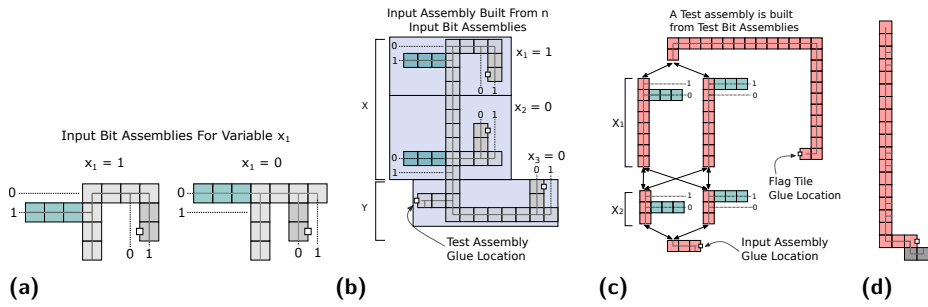
We now utilize covert computation to show that the open problem of Unique Assembly Verification in staged self assembly is PSPACE-complete. We start by showing UAV with 3 stages is Π_2^P -hard. We then show how to extend this construction to show that general staged UAV is PSPACE-complete. With some adjustments the same concept is used to show that when limiting the system to n stages, the problem of UAV is Π_{n-1}^P -hard.

► **Problem 2** (Staged Unique Assembly Verification). *Given a staged system Γ and an assembly A , does Γ uniquely assemble A ?*

4.1 3-stage UAV is Π_2^P -hard

We modify the covert computation construction to provide a reduction from $\forall\text{ESAT}$. Given an instance of $\forall\text{ESAT}$, we create a 3-stage temperature-2 staged system that uniquely produces a target assembly iff the given instance of $\forall\text{ESAT}$ is true. The reduction uses the same high-level idea as [16] and [3]. The process begins with the construction of an assembly for every input to the $\forall\text{ESAT}$ formula. Circuit assemblies build from these inputs and are flagged as true or false, while encoding a partial assignment through their geometry. Separate “test” assemblies are constructed that also encode a partial assignment to the same variables, which attach to true circuit assemblies with matching assignments. The systems uniquely assembles a target assembly if for all test assemblies there exists a compatible true circuit assembly for it to attach to. See Figure 6 for a visual overview of the created system.

► **Problem 3** ($\forall\text{ESAT}$). *Given an n -bit boolean formula $\phi(x_1, x_2 \dots x_n)$ with the inputs divided into two sets X and Y , for every assignment to X , does there exist an assignment to Y such that $\phi(X, Y) = 1$?*



■ **Figure 7** (a) Input bit assemblies for variables in X with geometry on the left reflecting the bit value. (b) An example initial circuit assembly for input $x_1 = 1, x_2 = 0, x_3 = 0$. The geometry on the left side of the assembly represents the assignment of X . (c) Separately built test bit assemblies nondeterministically attach to build one test assembly for every assignment to variables in X . (d) The blank test assembly is composed of the same base but has no protruding arms.

4.1.1 First Stage

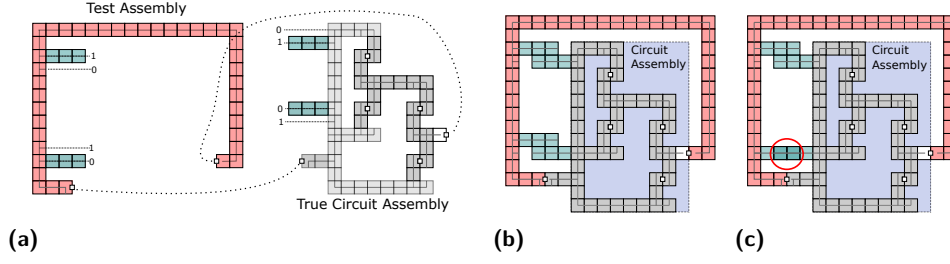
Input and Gate Assemblies. In the first stage an *input bit assembly* for both assignments to every variable x_1, \dots, x_n is built in its own bin ($2n$ bins in total). Input bit assemblies have the same structure as in the covert computation construction, except that input bit assemblies representing bits in X also have a horizontal row of tiles on the left of the frame that reflects the bit value. Figure 7a shows this modification to the input bit assemblies. The bit assemblies representing variable x_n no longer has additional tiles that attach to the test tile used in section 3. The input bit assemblies representing variable $x_{|X|+1}$ have an additional 2 tiles attached, which are used to attach to the test assembly. Gate Assemblies are built in the same way described in Section 3.

Test Assemblies. Similar to the input bit assemblies, two *test bit assemblies* are constructed for every variable in X . A test bit assembly is a column of connected tiles, with a horizontal row of 3 tiles extending to the right, the position of this row represents an assignment “0” or “1”. An example test assembly building from separate test bit assemblies is shown in Figure 7c. A test assembly is composed of $|X|$ test bit assemblies. Test assemblies have additional geometry that allow them to attach to a circuit assembly.

4.1.2 Second Stage

In the second stage the input bit assemblies will attach together nondeterministically to form 2^n unique input assemblies. The “1” and “0” input bit assembly exist for every variable, so the nondeterministic nature of the model allows for the construction of an input assembly for every possible input to the circuit. From this input assembly, computation will begin as described in the covert section. There will exist a circuit assembly for each of the 2^n possible inputs, and each will be flagged as true or false, represented by the existence of a flag tile on the output gate. We call a circuit assembly that contains the flag tile a *true circuit assembly*.

Test Assemblies. Test bit assemblies nondeterministically combine in this stage to create a unique test assembly for every assignment to variables in X , with its assignment encoded in its geometry. A test assembly can cooperatively bind to a true circuit assembly (with the same assignment to X) by having glue strength with the output flag tile and the input assembly (Figure 8b). A test assembly has its assignment encoded in its geometry in a



■ **Figure 8** (a) A test assembly (left) and a true circuit assembly that represent the same assignment to variables in X (In this construction true assemblies contain the flag tile). (b) A test assembly attaching to true circuit assembly with a matching assignment to X . (c) A test assembly that is geometrically blocked from attaching to a true circuit assembly due to having a different assignment to X . The red circled area shows the point of overlap.

complementary fashion to that of a circuit assembly. This ensures that a test assembly is geometrically blocked from attaching to a circuit assembly that encodes a different assignment to X (Figure 8c). If there are no true circuit assemblies with the assignment x to X , the test assembly that represents that assignment of x will be terminal in the second stage. We refer to these as *unmatched* test assemblies.

► **Lemma 4.** *Let t_x be the test assembly representing the assignment x to the variables in X . t_x is unmatched (terminal in the second stage) if and only if for all assignments y to the variables in Y ($\phi(x, y) = 0$).*

Proof. True circuit assemblies are the only assemblies which have the necessary glue types to attach to t_x . The remaining question is whether a true circuit assembly exists which represents a compatible assignment. t_x can attach to any true circuit assembly with a matching assignment to X , regardless of that circuit's assignment to Y . It follows that if there exists an assignment y to Y such that $\phi(x, y) = 1$, then t_x is not terminal. The negation of which is $\forall y(\phi(x, y) = 0)$, then t_x is terminal. ◀

4.1.3 Third Stage

The third stage utilizes a single bin that all assemblies are combined in. Nearly all single tiles of the target assembly are added. Four single tiles are specifically excluded, and instead two subassemblies are added in. This is done carefully to ensure the following property: every assembly except unmatched test assemblies from the second stage will grow to the target assembly. Our target assembly contains a circuit assembly attached to a test assembly with every empty spot filled in. At the point where a test assembly attaches to the circuit assembly, a domino assembly is attached completing the target assembly as seen in Figure 9a.

► **Lemma 5.** *Let A be the set of initial assemblies in the sole bin in the third stage. For all assemblies $a \in A$, a will grow to the target assembly iff a is not an unmatched test assembly.*

Proof. All individual tiles of the target assembly are added into the last stage, with the exception of four withheld tiles: the two tiles where the test assembly and input assembly meet, and the two tiles below that (tiles A, B, C, D in Fig. 9a). Instead of these four tiles, two assemblies are added that we refer to as the square and domino (Fig. 9b). These two assemblies perform the function of allowing every initial assembly besides unmatched test assemblies to grow into the target assembly. True circuit assemblies with test assemblies attached will have their empty spaces filled by single tiles, and the domino assembly will

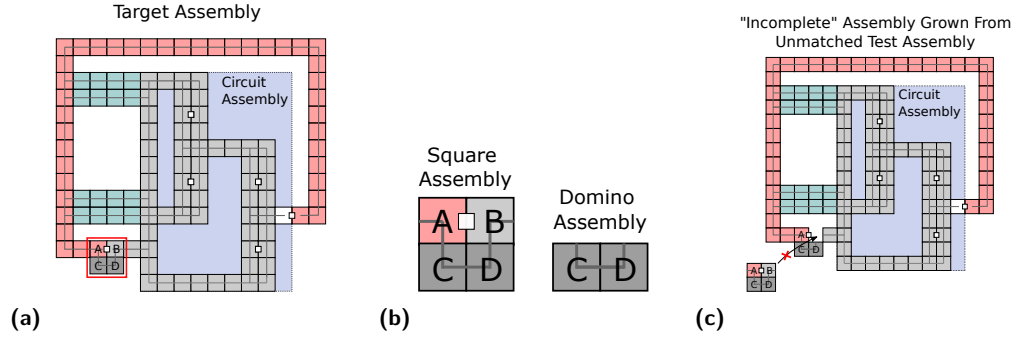


Figure 9 (a) An example target assembly. The area boxed in red shows where the test assembly meets the input assembly (A, B) , and the adjacent domino (C, D) . The four tile types A, B, C, D are not added in individually at the third stage. (b) The two additional assemblies that are added in at the third stage, composed of the same four tile types. (c) The square assembly is geometrically blocked from attaching to an assembly that grew from an unmatched test assembly, as they both contain tile A .

attach. Unused gates will grow to a near-complete circuit, attach to the square assembly, and then continue to grow to the target assembly. True and False Circuit Assemblies with blank test assemblies attached already contain the four withheld tiles, so will grow to the target by attaching to all necessary single tiles. Unmatched test assemblies that did not attach to a true circuit assembly can grow to a near complete target assembly, however, it will never acquire tile B (Fig. 9a), as it could only achieve this by attaching to the square assembly. They both contain tile A , making it geometrically blocked from doing so (Fig. 9c). ◀

► **Theorem 6.** *UAV in the Staged Assembly Model with three stages is Π_2^P -hard with $\tau = 2$.*

Proof. Given an instance of $\forall\text{ESAT}$, the reduction provides an instance of a 3-stage temperature-2 UAV instance which is true if and only if the instance of $\forall\text{ESAT}$ is true.

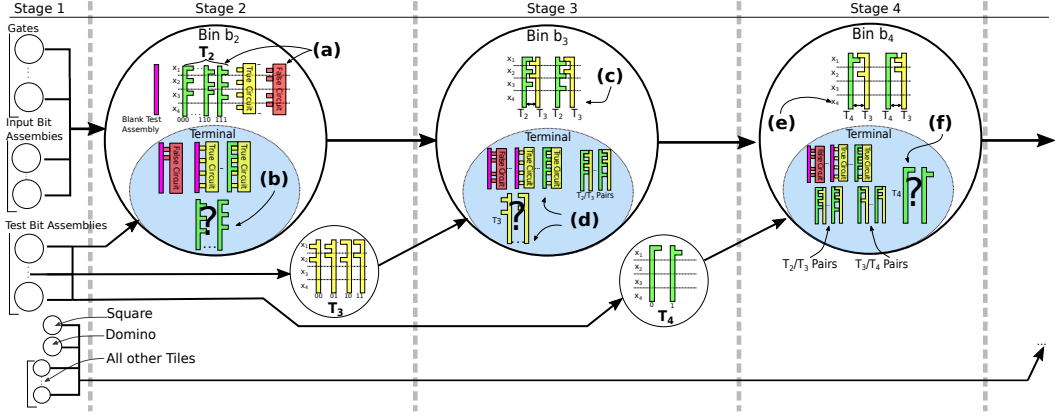
If the instance of $\forall\text{ESAT}$ is true, then for all assignments x to X , there exists an assignment y to Y with $\phi(x, y) = 1$. By Lemma 4, this implies there will be no unmatched test assemblies. By Lemma 5, every assembly that is not an unmatched test assembly or grown from an unmatched test assembly will grow into the target assembly in the third stage. Thus, the system uniquely produces the target assembly. If the $\forall\text{ESAT}$ instance is false, then there exists an assignment x to X , s.t. for all assignments y to Y , $\phi(x, y) = 0$. By Lemma 4, a test assembly representing assignment x would be unmatched, and by Lemma 5, unable to grow into the target assembly. Thus, this UAV instance is false. ◀

4.2 Staged UAV is PSPACE-hard

In this section, we explain at a high level how the reduction is extended to reduce from TQBF with n quantifiers over n variables to temperature-2 $\mathcal{O}(n)$ -stage UAV, showing that Staged UAV is PSPACE-Hard.

► **Problem 7 (TQBF).** *Given a boolean formula ϕ with n variables x_1, \dots, x_n , is it true that $\forall x_1 \exists x_2 \dots \forall x_n (\phi(x_1, \dots, x_n) = 1)$?*

We utilize the same technique used in section 4.1 which reduced from $\forall\text{ESAT}$, a special case of TQBF limited to only 2 quantifiers, but adapt the technique to work with a QBF with n quantifiers $\forall x_1 \exists x_2 \dots \forall x_n (\phi(x_1, \dots, x_n) = 1)$. In the 3rd stage, instead of adding



■ **Figure 10** An example mix graph for an instance of TQBF with 4 variables. (a) Test bit assemblies combine into T_2 test assemblies. Circuit assemblies evaluate every input. (b) T_2 test assemblies attach to compatible (matching partial assignment) true circuits. Any unmatched T_2 assemblies are passed to the next stage. (c) T_3 test assemblies are added in and attach to compatible T_2 test assemblies. (d) Any unmatched T_3 assemblies are passed to the next stage. (e) T_4 test assemblies are added and attach to compatible T_3 test assemblies. (f) The existence of an unmatched T_4 assembly directly corresponds to the truth of the TQBF instance.

in single tiles to “clean up”, we add in a second set of test assemblies that represent an assignment with one less variable in the next stage and are complementary in their geometry. These new test assemblies then attach to previous test assemblies that were terminal in the previous stage with matching partial assignments. This process computes an additional quantifier. We can then repeat this process of adding in complementary sets of test assemblies for the number of quantifiers required. In the final stage, if a test assembly from the final set couldn’t find a complementary test assembly to attach to, the instance of TQBF is false, and that test assembly is prevented from growing to the target assembly. This allows the truth of instance of staged UAV to correspond to the truth of the QBF. See Figure 10 for a depiction of the mix graph. We now show how in a certain stage the existence of a terminal test assembly relates to the truth of a statement about the boolean formula.

In total the system will have $n + 1$ stages, and $n - 1$ sets of test assemblies will be added (denoted T_2, \dots, T_n). The set T_s will be mixed in at stage s . The first set T_2 represents an assignment to x_1, \dots, x_{n-1} , and each consecutive set represents one less variable than the set before it, i.e., a test assembly $t_s \in T_s$ represents a partial assignment to x_1, \dots, x_{n-s-1} . The sets alternate between type L and R , which correlates to the direction the arms face (Compare T_3 and T_4 in Figure 10). We build all these sets of test assemblies using the same method in the first stage, and pass them along through “helper” bins until they are needed.

► **Lemma 8.** Let $TERM(A, b) \iff (\text{Assembly } A \text{ is terminal in bin } b)$. Let a be the number of variables the test assemblies in T_s represent ($a = n - s + 1$). Let $t_s(x_1, \dots, x_a)$ be the test assembly $t_s \in T_s$ that represents partial assignment x_1, \dots, x_a . In the staged system S_P created from an instance of TQBF P over n variables: $\forall s \in \{1, \dots, n\} (TERM(t_s(x_1, \dots, x_a), b_s) \iff \forall x_{a+1} \exists x_{a+2}, \dots, Qx_n (\phi(x_1, \dots, x_n) = y))$. If s is even, $y = 0$ and $Q = \forall$, and $y = 1$, $Q = \exists$ otherwise.

► **Lemma 9.** In the staged system S_P created from an instance of TQBF P over n variables, in bin b_{n+1} in stage $n + 1$, let A be the set of initial assemblies in b_{n+1} . For all $a \in A$, a will grow to the target assembly if and only if a is not an unmatched test assembly $t_n \in T_n$.

► **Theorem 10.** *Unique Assembly Verification in the Staged Assembly Model is PSPACE-complete with $\tau = 2$.*

Proof. Given an instance of TQBF P over n variables/quantifiers, the reduction provides an instance of $n + 1$ -stage $\tau = 2$ UAV that is true if and only if P is true. If P is true, then in stage $n + 1$, every producible assembly grows into the target assembly. Since n is always even, by Lemma 8, for a bin b_n in stage n , an assembly $t_n \in T_n$ representing an assignment x_1 is terminal in bin b_n if $\forall x_2 \exists x_3, \dots, \forall X_n (\phi(x_1, \dots, x_n) = 0)$. If P is true, then the statement $\forall x_1 \exists x_2 \forall x_3, \dots, \forall X_n (\phi(x_1, \dots, x_n) = 1)$ is true, and therefore no unmatched $t_n \in T_n$ will be passed into b_{n+1} . By Lemma 9, every initial assembly in b_{n+1} that is not some $t_n \in T_n$ grows into the target assembly. Therefore, the target assembly is uniquely assembled if the instance of TQBF is true. If P is false, then there exists an assignment to x_1 such that $\forall x_2 \exists x_3, \dots, \forall X_n (\phi(x_1, \dots, x_n) = 0)$. By Lemma 8, some test assembly $t_n \in T_n$ will be terminal and passed into bin b_{n+1} . By Lemma 9, any $t_n \in T_n$ will not grow into the target assembly. Thus, the instance of staged UAV is false. ◀

4.3 n -Stage Hardness

We now show how the reduction can be used to show hardness for n -stage UAV. We reduce from the boolean satisfiability problem for Π_n^p , which is a quantified boolean formula with n quantifiers (starting with universal) and $n - 1$ alternations. We show an instance of Π_n^p -SAT can be reduced to $n + 1$ -stage $\tau = 3$ UAV.

► **Problem 11** (Π_n^p - SAT). *Given a boolean formula ϕ with variables partitioned into n sets X_1, \dots, X_n , is it true that $\forall X_1 \exists X_2 \dots Q_n X_n (\phi(X_1, \dots, X_n))$.*

► **Theorem 12.** *For all $n > 1$, UAV in the Staged Assembly Model with n stages is Π_{n-1}^p -hard with $\tau = 2$.*

Proof. The system functions nearly identically to the previous reduction. However, if n is odd, the output gate assemblies will now contain the flag tile if they represent a *false* output, rather than true. Each consecutive test assembly added now represents one less *set* of variables, rather than just one less variable.

If n is even, the system acts in the way previously described. If n is odd, then by Lemma 8 any $t_n \in T_n$ representing an assignment to X_1 is terminal if $\forall X_2 \exists X_3 \dots \exists X_n (\phi(X_1, \dots, X_n) = 1)$. However, since we modified the output assemblies to contain the flag tile if they represent a *false* output, they are now terminal if the statement is true for the negation of ϕ . Therefore any t_n representing X_1 is terminal if and only if $\forall X_2 \exists X_3 \dots \exists X_n (\phi(X_1, \dots, X_n) = 0)$. In bin b_{n+1} all assemblies besides any t_n grow to the target assembly in the same way. ◀

4.4 UAV Membership

In this section, we improve on previous work and show that an n -stage UAV problem is in Π_{n+1}^p . We use a similar method as [16], by defining three subproblems that are solved as subroutines of a UAV algorithm. However, these subproblems differ from previous work as we make some assumptions about our input. We first define *bounded* bins and systems, then define the three subproblems, and show their complexity. However, due to space constraints, the proofs have been omitted.

► **Definition 13** (Bounded). *Given a bin $b = (S, \tau)$ in a staged system where S is the set of initial assemblies and τ is the temperature. Let P_b be the set of producible assemblies in bin b . The bin is bounded by an integer $k \in \mathbb{Z}^+$ if for each $a \in P_b$, $|a| \leq k$. A staged system is bounded if all bins are bounded by some k .*

■ **Table 2** Complexity of these problems in 1 stage (2HAM) and in s stages.

Stages	UAV	BPROD	BTERM	BBIN
1	Π_1^p	Σ_0^p	Π_1^p	Π_1^p
s	Π_{s+1}^p	Σ_s^p	Π_s^p	Π_s^p

■ **Algorithm 1** Staged Unique Assembly Verification Membership Algorithm.

Data: Given a staged system Γ with n stages, and an Assembly A .

Result: Does Γ uniquely assemble A and is Γ bounded?

for each stage s' starting with $s' = 1$ **do**

for each bin b in stage s' **do**

if Not $BBIN_{s'}(\Gamma, |A|, b')$ **then** reject;

for each bin b in stage n **do**

if Not $BPROD_n(\Gamma, |A|, b, A)$ **then** reject;

if Not $BTERM_n(\Gamma, |A|, b, A)$ **then** reject;

Nondeterministically select an assembly B with $|B| \leq |A|$;

for each bin b' in stage n **do**

if $BPROD_n(\Gamma, |A|, b', B)$ **then**

if $BTERM_n(\Gamma, |A|, b', B)$ **then** reject;

accept;

► **Problem 14** (Bounded Producibility ($BPROD_s$)). *Given a bounded staged system Γ , an integer k (described in unary), a bin b in stage s bounded by k , and an assembly A , is A producible in b ?*

► **Problem 15** (Bounded Terminal Assembly with producibility promise ($BTERM_s$)). *Given a bounded staged system Γ , an integer k (described in unary), a bin b in stage s bounded by k , and an assembly $A \in P_b$, is A terminal in b ?*

► **Problem 16** (Bounded Bin ($BBIN_s$)). *Given a staged system Γ , a bin b in stage s , an integer k (described in unary), assuming all bins in stages before s are bounded by k , is b bounded by k ?*

► **Lemma 17.** *For a bin b in stage s of a staged self-assembly system,*

■ *the Bounded Producibility problem is in Σ_s^p ,*

■ *the Bounded Terminal Assembly problem with producibility promise is in Π_s^p , and*

■ *the Bounded Bin problem is in Π_s^p*

4.5 UAV_n Membership

We now present a co-nondeterministic algorithm using oracles for the previous problems to solve UAV. For clarity, we use an alternate but equivalent definition of UAV. We provide Algorithm 1 that uses oracles to solve the subproblems presented above.

► **Problem 18** (Staged Unique Assembly Verification). *Given a staged tile-assembly system Γ and an assembly A , is Γ bounded by $|A|$, and for each bin in the last stage, is A the only terminal assembly?*

► **Theorem 19.** *The n -stage Unique Assembly Verification problem in the staged assembly model is in Π_{n+1}^p .*

5 Conclusion

In this paper we answered an open problem from [16] by showing the Unique Assembly Verification problem in the Staged Self-Assembly Model is PSPACE-complete. To show this, we utilized a construction capable of covert computation and extended it to show Π_2^P -hardness of UAV with three stages. We then extended this reduction to show PSPACE-completeness. This reduction is also used to show Π_{s-1}^P -hardness with s stages.

Several important directions for future work remain open. We use three stages to perform covert computation. Is the 2HAM alone capable of covert computation? If not, what is the lower bound on the number of stages needed? If so, can the construction be used to solve the open problem of UAV in that model? This might also mean fewer stages are needed for our results in the staged model. The two known hardness results for 2HAM utilize either one step into the third dimension or a variable temperature. Perhaps stronger results in the staged assembly model can be obtained with one of these variants.

References

- 1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert Schweller. Shape replication through self-assembly and rnaase enzymes. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1045–1064, 2010. doi:10.1137/1.9781611973075.85.
- 2 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothmund. Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.
- 3 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Verification and Computation in Restricted Tile Automata. In Cody Geary and Matthew J. Patitz, editors, *26th International Conference on DNA Computing and Molecular Programming (DNA 26)*, volume 174 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.2020.10.
- 4 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M Summers, and Andrew Winslow. Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 172–184. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.
- 5 Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. *Algorithmica*, 83:531–552, 2021. arXiv:1908.06068. doi:10.1007/s00453-020-00764-w.
- 6 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018. doi:10.1007/s00453-017-0318-0.
- 7 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of linear assemblies. *Natural Computing*, 18(3):527–548, 2019. doi:10.1007/s11047-019-09740-y.
- 8 Erik Demaine, Matthew Patitz, Robert Schweller, and Scott Summers. Self-assembly of arbitrary shapes using rnaase enzymes: Meeting the kolmogorov bound with small scale factor. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, January 2010. doi:10.4230/LIPIcs.STACS.2011.201.

- 9 Erik D Demaine, Martin L Demaine, Sándor P Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T Schweller, and Diane L Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $o(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.
- 10 Erik D. Demaine, Sarah Eisenstat, Mashhood Ishaque, and Andrew Winslow. One-dimensional staged self-assembly. In *Proceedings of the 17th international conference on DNA computing and molecular programming*, DNA’11, pages 100–114, 2011.
- 11 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017. Computational Self-Assembly. doi:10.1016/j.tcs.2016.11.020.
- 12 David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013.
- 13 Alexandra Keenan, Robert Schweller, Michael Sherman, and Xingsi Zhong. Fast arithmetic in algorithmic self-assembly. *Natural Computing*, 15(1):115–128, March 2016.
- 14 Matthew Patitz and Scott Summers. Identifying shapes using self-assembly. *Algorithmica*, 64:481–510, 2012.
- 15 Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.
- 16 Robert Schweller, Andrew Winslow, and Tim Wylie. Verification in staged tile self-assembly. *Natural Computing*, 18(1):107–117, 2019.
- 17 Grigory Tikhomirov, Philip Petersen, and Lulu Qian. Fractal assembly of micrometre-scale dna origami arrays with arbitrary patterns. *Nature*, 552, December 2017. doi:10.1038/nature24655.
- 18 Andrew Winslow. Staged self-assembly and polyomino context-free grammars. *Natural Computing*, 14(2):293–302, 2015. doi:10.1007/s11047-014-9423-z.
- 19 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable dna self-assembly. *Nature*, 567:366–372, March 2019. doi:10.1038/s41586-019-1014-9.

An Instance-Optimal Algorithm for Bichromatic Rectangular Visibility

Jean Cardinal  

Université libre de Bruxelles (ULB), Brussels, Belgium

Justin Dallant 

Université libre de Bruxelles (ULB), Brussels, Belgium

John Iacono  

Université libre de Bruxelles (ULB), Brussels, Belgium

Abstract

Afshani, Barbay and Chan (2017) introduced the notion of instance-optimal algorithm in the order-oblivious setting. An algorithm A is instance-optimal in the order-oblivious setting for a certain class of algorithms \mathcal{A} if the following hold:

- A takes as input a sequence of objects from some domain;
- for any instance σ and any algorithm $A' \in \mathcal{A}$, the runtime of A on σ is at most a constant factor removed from the runtime of A' on the worst possible permutation of σ .

If we identify permutations of a sequence as representing the same instance, this essentially states that A is optimal on every possible input (and not only in the worst case).

We design instance-optimal algorithms for the problem of reporting, given a bichromatic set of points in the plane S , all pairs consisting of points of different color which span an empty axis-aligned rectangle (or reporting all points which appear in such a pair). This problem has applications for training-set reduction in nearest-neighbour classifiers. It is also related to the problem consisting of finding the decision boundaries of a euclidean nearest-neighbour classifier, for which Bremner et al. (2005) gave an optimal output-sensitive algorithm.

By showing the existence of an instance-optimal algorithm in the order-oblivious setting for this problem we push the methods of Afshani et al. closer to their limits by adapting and extending them to a setting which exhibits highly non-local features. Previous problems for which instance-optimal algorithms were proven to exist were based solely on local relationships between points in a set.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases computational geometry, instance-optimality, colored point sets, empty rectangles, visibility

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.24

Related Version *Full Version:* <https://arxiv.org/abs/2106.05638> [12]

Funding *Justin Dallant:* This work was supported by the French Community of Belgium via the funding of a FRIA grant.

John Iacono: Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.

1 Introduction

In the theoretical study of algorithms one often quantifies the performance of an algorithm in terms of the worst case or average running time over a distribution of inputs of a given size. Sometimes, more precise statements can be made about the speed of an algorithm on certain instances by expressing the running time in terms of some parameter depending on the input. One class of such algorithms are the so-called output-sensitive algorithms, where the parameter is the size of the output. In computational geometry, a classical example is



© Jean Cardinal, Justin Dallant, and John Iacono;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 24; pp. 24:1–24:14

Leibniz International Proceedings in Informatics



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

computing the convex hull of a set of n points in the plane in $\mathcal{O}(n \log h)$ time, where h is the size of the convex hull [15, 5]. More recently, Afshani et. al. [1] introduced a specific notion of *instance optimality in the order-oblivious setting* and designed algorithms with this property for different problems on point sets. Roughly speaking, an algorithm is instance optimal in a certain class of algorithms \mathcal{A} if on any input sequence S , its performance on S is at most a constant factor removed from the best performance of any algorithm in \mathcal{A} on S . In the order-oblivious setting, performance is defined as the worst-case runtime over all permutations of the input sequence (the motivation behind this will be made clear below). A common characteristic of most problems solved in [1] is that they are based around local relations between points in S , in the sense that the relation between two points $p, q \in S$ depends solely on their coordinates and not on those of any other point in S . This is important because it allows one to decompose certain queries into independent queries on a partition of S . In this paper we push these methods closer to their limits by adapting and applying them to a problem which does not fit in this framework.

The studied problem. Here we consider the following problem: given a set S of points n in the plane with distinct x and y coordinates, each colored red or blue, report the red/blue pairs of points such that the rectangles they span contain no point of S in their interior. This again has applications to machine learning and more specifically nearest-neighbour classifiers. Indeed, by solving this problem and discarding all points which do not appear in such a pair, we obtain a (possibly much smaller) set of points where for any point y in the plane, the color of its nearest neighbour is the same as in the original set of points for the L_1 distance (as pointed out in [10], where this problem was perhaps first studied). This remains true even when *a priori* unknown (and possibly non-linear) scalings might be applied to the x and y axis after this preprocessing step. In fact, the resulting set of points consists of exactly those necessary for this to hold, so this constitutes an optimal reduction of the training set in that sense. Note that this problem does not fit in the general framework of [1], as whether two point span an empty rectangle depends on the position of all other points.

Related works. Pairs of points spanning empty rectangles and the corresponding graphs have been studied at numerous occasions in the past, under various names. They are called *rectangular influence graphs* in [10], which discusses applications to data clustering and classification. In [9] a similar relation is called *direct dominance*, and a worst-case optimal algorithm to report all pairs of related points is given. This algorithm runs in $\mathcal{O}(n \log n + h)$ time, where h is the size of the output. A straightforward adaptation yields an algorithm with the same running time for the bichromatic problem studied here. In [18] this relation is called *rectangular visibility* and a different algorithm with the same running time is given as well as algorithms for the dynamic query setting. The expected size of a largest independent set in this graph is studied in [7] (where they call such graphs *Delaunay graphs with respect to rectangles*). Generalizations and variations of this type of relation between points have also been widely studied [17, 16, 4, 2, 11, 14, 13, 19, 8]. Another problem of note which is closely related to the one we study here is the following: given a set S of n points in the plane, each colored red or blue, compute the subset of edges of the Voronoï diagram of S which are adjacent to both a site corresponding to a blue point and a site corresponding to a red point. This problem has some relevance to machine learning as we can equivalently state it as finding the boundaries of a nearest-neighbour classifier with two classes in the plane. A third formulation is finding the pairs of red and blue points such that there is an empty disk whose boundary passes through both. In [3], Bremner et. al. show that this problem can

be solved in output-sensitive optimal $\mathcal{O}(n \log h)$ time, where h is the size of the output. It is an interesting open problem to find instance-optimal algorithms for this problem in the order-oblivious setting (or prove that no such algorithm exists).

Paper organization. In Section 2 we motivate and state more precisely the notion of instance-optimality we work with in this paper. In Section 3 we define the problem formally and give an instance lower bound in the order-oblivious model by adapting the adversarial argument of [1]. The key new ingredients are a new definition of safety and a way to deal with the fact that here the adversary cannot necessarily change the expected output of the algorithm by moving a single point inside a so-called non-safe region. In Section 4 we give an algorithm and prove that its runtime matches the lower bound. The main observation which makes this work is that while the algorithms in [1] require the safety queries to be decomposable (which they are not here), we can afford to do some preprocessing to make them behave as if they were decomposable, as long as the amount of work done stays within a constant of the lower bound. In section 5 we mention that when competing against algorithms which can do linear queries, instance-optimality in the order-oblivious setting is impossible. Some details and proofs have been left out of this version and can be found in the full paper [12].

2 Instance optimality in the order-oblivious setting and model of computation

Ideally, we would like to consider a very strong notion of optimality, where an algorithm is optimal if on every instance its runtime is at most a constant factor removed from the algorithm with the smallest runtime on that particular instance. There is an obvious flaw with this definition, as for every instance we can have an algorithm “specialized” for that instance, which simply checks if the input is the one it is specialized for then returns the expected output without any further computation when it is the case (and spends however much time it needs to compute the correct output otherwise). For problems which are not solvable in linear time in the worst-case, this prohibits the existence of such optimal algorithms. One way to get around this issue in some cases and get a meaningful notion of instance-optimality is the following, taken from [1].

► **Definition 1.** *Consider a problem where the input consists of a sequence of n elements from a domain \mathcal{D} . Consider a class \mathcal{A} of algorithms. A correct algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in \mathcal{D} . For a set S of n elements in \mathcal{D} , let $T_A(S)$ denote the maximum running time of A on input σ over all $n!$ possible permutations σ of S . Let $\text{OPT}(S)$ denote the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$. If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \cdot \text{OPT}(S)$ for every set S , then we say A is instance-optimal in the order-oblivious setting.*

By measuring the performance of an algorithm on an instance as the maximum runtime over all permutations of the instance elements, the algorithm can no longer take advantage of the order in which the input elements are presented. In particular, simply checking if the input is a specific sequence is no longer a good strategy. Here, the domain \mathcal{D} consists of all points in the plane, colored red or blue. An instance is a sequence of points, no two sharing the same x or y coordinate. However, we really want to consider this sequence as a set of points, as the order in which the points are presented does not change the instance conceptually. Thus, it makes sense for us to consider a performance metric for which algorithms cannot

take advantage of this order. For the class of algorithms \mathcal{A} , we will consider algorithms in a restricted real RAM model where the input can only be accessed through comparison queries. That is, the algorithms can compare the x or y coordinates of two points but not, for example, evaluate arbitrary arithmetic expressions on these coordinates. We refer to such algorithms as *comparison-based algorithms*. The lower bound works even for a stronger model of computation, comparison-based decision trees (assuming at least a unit cost for every point returned in the output). We could also allow the comparison of the x coordinate of a point with the y coordinate of another without changing any of the results of this paper.

3 Lower-bound for comparison-based algorithms in the order-oblivious setting

Some basic notations and definitions

Throughout this section we consider a set S of n red and blue points in the plane. We assume that S is non-degenerate, in the sense that no two points in S share the same x or y coordinate (in particular, all points are distinct). If p is a point, we will denote its x and y coordinates as $x(p)$ and $y(p)$ respectively and its color as $c(p)$.

► **Definition 2.** A point p dominates $q \neq p$ if $x(p) \geq x(q)$ and $y(p) \geq y(q)$. A point is maximal (resp. minimal) in S if it is dominated by (resp. dominates) no point in S .

► **Definition 3** (Visible and participating points). Let p, q be two points in S . We say that q is visible from p in S (or that p sees q in S) if the axis-aligned box spanned by p and q contains no point of S in its interior. We say that $p \in S$ participates in S if it is visible from a point in S of the opposite color. We will omit the set S when it is clear from context.

The problems we want to solve can thus be restated as follows.

► **Problem 4** (Reporting participating points). Report all points which participate in S .

► **Problem 5** (Reporting red-blue pairs of visible points). Report all red-blue pairs of points (p, q) such that p and q see each other in S .

The following definitions will also be useful for us.

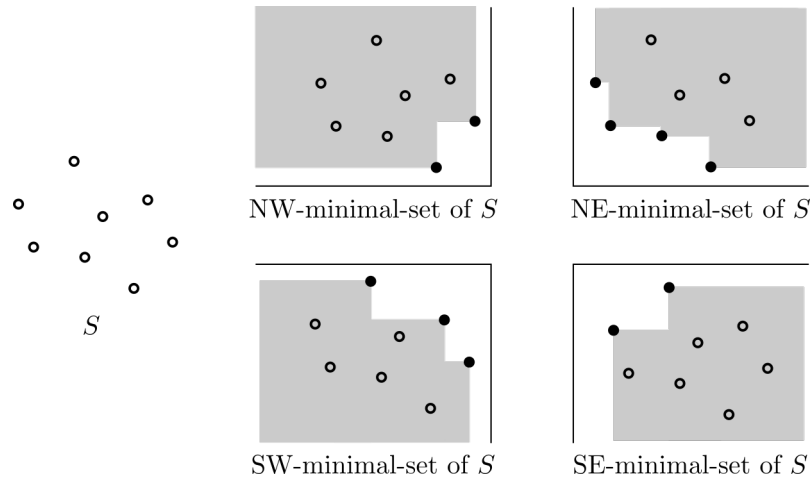
► **Definition 6.** We call the set of minimal points of S the NE-minimal-set of S (for “North-East-minimal set”). The NW-minimal-set, SE-minimal-set and SW-minimal-set of S are defined symmetrically (see Figure 1). In particular, the SE-minimal set of S is the set of maximal points in S .

► **Definition 7.** Let B be an axis-aligned box. We denote the x coordinate of the right boundary (resp. left boundary) of B as $x_{\max}(B)$ (resp. $x_{\min}(B)$). We denote the y coordinate of the top boundary (resp. bottom boundary) of B as $y_{\max}(B)$ (resp. $y_{\min}(B)$).

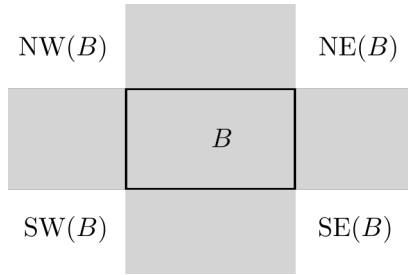
The cross of B , denoted as $\text{cross}(B)$ is the set of points p in the plane such that $x_{\min}(B) \leq x(p) \leq x_{\max}(B)$ or $y_{\min}(B) \leq y(p) \leq y_{\max}(B)$.

The quadrants of B are the connected components of $\mathbb{R}^2 \setminus \text{cross}(B)$. We call the four components the NE-quadrant, NW-quadrant, SE-quadrant and SW-quadrant, denoted as $\text{NE}(B)$, $\text{NW}(B)$, $\text{SE}(B)$ and $\text{SW}(B)$ respectively (see Figure 2).

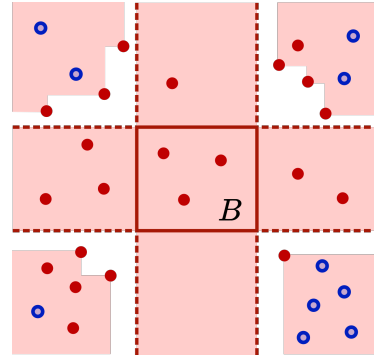
All boxes we consider are axis-aligned boxes in the plane. For ease of exposition, we assume that all boxes we consider have no point of S on the boundaries of their four quadrants.



■ **Figure 1** A set of points S and the four minimal-sets of S . No point in the shaded regions can appear on the corresponding minimal-sets.



■ **Figure 2** The four quadrants of an axis-aligned box B . The shaded region corresponds to $\text{cross}(B)$.



■ **Figure 3** A red-safe box B .

Lower Bound

We prove an entropy-like lower bound on the number of comparisons which have to be done to solve the problems of reporting participating points in the order-oblivious setting. The proof and terminology are largely inspired from the lower bound proofs in [1] but some additional arguments, which we underline later, are required. We need a few definitions in order to state our lower bound.

► **Definition 8.** *An axis-aligned box B is red-cross-safe (resp. blue-cross-safe) for S if all points in $S \cap \text{cross}(B)$ are red (resp. blue). It is cross-safe if it is red-cross-safe or blue-cross-safe.*

It is red-safe if it is red-cross-safe and the NE-minimal (resp. NW-minimal, SE-minimal, SW-minimal) set of $S \cap \text{NE}(B)$ (resp. $S \cap \text{NW}(B)$, $S \cap \text{SE}(B)$, $S \cap \text{SW}(B)$) is red (see Figure 3). We define blue-safe boxes similarly. A box is safe if it is red-safe or blue-safe.

A subset $S' \in S$ has one of these properties if it can be enclosed by a box with the property.

Notice that if a subset $S' \in S$ is safe, then no point in S' participates in S . Thus, in an intuitive sense, a partition of non-participating points into safe subsets can be seen as a certificate for the fact that these points do not participate. The minimal entropy of such a partition is then the minimal amount of information required to encode such a certificate.

► **Definition 9.** A partition Π of S is *respectful* if every member $S_k \in \Pi$ is a singleton or a safe subset of S . The entropy of a partition Π is $\mathcal{H}(\Pi) := \sum_{S_k \in \Pi} \frac{|S_k|}{n} \log \frac{n}{|S_k|}$. The structural entropy $\mathcal{H}(S)$ of S is the minimum of $\mathcal{H}(\Pi)$ over all respectful partitions Π of S .

We can now state our lower bound:

► **Theorem 10.** For the problem of reporting participating points in the order-oblivious comparison-based model, $\text{OPT}(S) \in \Omega(n(\mathcal{H}(S) + 1))$. Consequently, for reporting red-blue pairs of visible points, $\text{OPT}(S) \in \Omega(n(\mathcal{H}(S) + 1) + h)$, where h is the size of the output.

The proof is similar to what can be found in [1]. In their case however, if a point can be moved anywhere inside a non-safe box then it can be moved in a way that changes the expected output of the algorithm. In our case, we can do something similar but sometimes need to move multiple points to affect the expected output (but a constant number are enough). We use a simple argument about the maximum number of chips which can be placed on a tree in a constrained way to show that this has no impact on the lower bound. The full proof, which is too long for this version of the paper, can be found in the full paper [12]. From this lower bound one can also deduce that the existence of an instance-optimal algorithm in the order-oblivious setting for reporting participating points implies the existence of such an algorithm for reporting red-blue pairs of visible points (using known results about worst-case optimal algorithms). Thus, we will focus on the former problem from now on.

4 Instance optimal comparison-based algorithm in the order-oblivious setting

In this section we present a comparison-based algorithm for reporting participating points with a runtime matching the lower bound we saw in the previous section (note that worst-case optimal $\mathcal{O}(n \log n + h)$ algorithms are easy to obtain by the method of [9]). Once again, the main algorithm will be very similar to what is done in [1], however their results do not directly apply here. The main difficulty in adapting their algorithm to our case is that the relation we consider here is not decomposable. More precisely, if we know that some point p does or does not participate in S' and S'' , we cannot use this to decide if p participates in $S' \cup S''$. The bulk of the work here will thus be to preprocess the set S in order to make the safety tests decomposable, while keeping our preprocessing time within $\mathcal{O}(n(\mathcal{H}(S) + 1))$.

4.1 The main algorithm

Before we go into detail about how to preprocess the points let us see how, if we can build the right data structure, we can use it to report the participating points in S in $\mathcal{O}(n(\mathcal{H}(S) + 1))$ time. We will need the following definition and observations:

► **Definition 11.** Let S be a set of red and blue points. A subset $S' \subset S$ conforms with S if it contains all points which participate in S .

► **Observation 12.** Let S be a set of red and blue points and let S' be a subset which conforms with S . Then an axis-aligned box is safe for S if and only if it is safe for S' . Moreover, a point participates in S if and only if it participates in S' .

► **Observation 13.** If B is a safe box for S , then any sub-box of B is safe for S .

► **Observation 14.** Let p be a point in S and let B be some axis-aligned box bounding p . If B is safe, then p does not interact in S .

We have the following algorithm and theorem, adapted from [1], where δ is a constant to be chosen later:

■ **Algorithm 1** Reporting participating points.

Input: A point set S of size n

- 1 Set $Q = S$.
- 2 **for** $j = 0, 1, \dots, \lfloor \log(\delta \log n) \rfloor$ **do**
- 3 Partition the points in Q using a kd-tree to get $r_j = 2^{2^j}$ subsets Q_1, \dots, Q_{r_j} of size at most $\lceil |Q|/r_j \rceil$, along with corresponding bounding boxes B_1, \dots, B_{r_j} .
- 4 **for** $i = 0, 1, \dots, r_j$ **do**
- 5 **if** B_i is safe for Q **then**
- 6 Prune all points in Q_i from Q .
- 7 Solve the reporting problem for the remaining set Q directly in $O(|Q| \log |Q|)$ time.

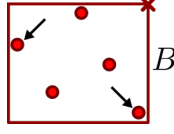
► **Theorem 15.** *Let S be a set of n points in general position. Suppose we have preprocessed S such that for any subset $S' \subset S$ containing all points which participate in S we can test if an axis-aligned box is safe for S' in $O(n^{1-\alpha})$ time, plus the cost of a constant number of range-emptiness queries on S' . Then Algorithm 1 can report all points which participate in S in $O(n(\mathcal{H}(S) + 1))$ time.*

We reiterate the proof for the sake of completeness and to underline how Observation 12 and our additional assumptions on preprocessing factor into it:

Proof. By Observation 12 and Observation 14, we only ever prune points which do not participate in the original set S and we never modify which points participate among those that remain. Thus the algorithm invoked at line 6 will compute the correct output.

By assumption, testing a box for safety in Q can be done in $O(n^{1-\alpha})$ time plus the cost of a constant number of range-emptiness queries on Q . Using a simple and ingenious trick by T. Chan [6], we can perform r orthogonal range emptiness queries on a set of size m in $O(m \log r + r^{O(1)})$ time. Thus, the r_j tests of lines 3 and 4 can be done in $O(|Q| \log r_j + r_j^{O(1)} + r_j n^{1-\alpha})$ time. As $r_j < n^\delta$, by taking δ small enough the $r_j^{O(1)} + r_j n^{1-\alpha}$ term can be made sublinear. As the outer loop of the algorithm is only executed $O(\log(\log n))$ times the total contribution of these terms over the whole algorithm can also be made sublinear and thus negligible. Line 2 can be done in $O(|Q| \log r_j)$ time by the classical recursive algorithm to compute kd-trees.

Now let n_j be the number of points in Q just after iteration j . The runtime of the algorithm is in $O(\sum_j n_j \log r_{j+1})$. This includes the final step at line 6, as for $j = \lfloor \log(\delta \log n) \rfloor$ (i.e. after the last iteration of the outer loop) we have $O(|Q| \log |Q|) \subset O(n_j \log n^{2\delta}) = O(n_j \log r_{j+1})$. Let Π be a respectful partition of S and consider $S_k \in \Pi$. At iteration j all subsets Q_i lying entirely inside the bounding box of S_k are pruned by Observation 13. Since the bounding box of S_k intersects at most $O(\sqrt{r_j})$ cells of the kd-tree, the number of points in S_k remaining after iteration j is $\min\{|S_k|, O(\sqrt{r_j} \cdot n/r_j)\} = \min\{|S_k|, O(n/\sqrt{r_j})\}$. The S_k 's cover the entire point set so by double summation we have:



■ **Figure 4** A cell of red points and the corresponding box B . The relevant points are indicated by arrows. The box point is indicated by a cross.

$$\begin{aligned}
 \sum_j n_j \log r_{j+1} &\leq \sum_j \sum_k \min \left\{ |S_k|, O \left(n / \sqrt{2^{2j}} \right) \right\} \cdot 2^{j+1} \\
 &= \sum_k \sum_j \min \left\{ |S_k|, O \left(n / 2^{2^{j-1}} \right) \right\} \cdot 2^{j+1} \\
 &\in O \left(\sum_k \sum_{j \leq \log(2 \log(n/|S_k|))} |S_k| \cdot 2^j + \sum_{j > \log(2 \log(n/|S_k|))} n \cdot 2^j / 2^{2^{j-1}} \right) \\
 &\in O \left(\sum_k |S_k| (\log(n/|S_k|) + 1) \right) = O(n(\mathcal{H}(S) + 1)). \quad \blacktriangleleft
 \end{aligned}$$

4.2 Cross-safety tree

In order to solve our problem instance-optimally (in the order-oblivious comparison-based model), we want to design a data-structure which allows us to quickly test if a given axis aligned box B is safe for S . To make the presentation clearer, we focus on testing if B is red-cross-safe and the NE-minimal-set of $\text{NE}(B) \cap S$ is red. This can then be repeated symmetrically for $\text{NW}(B)$, $\text{SE}(B)$ and $\text{SW}(B)$ to test if B is red-safe (and similarly for testing if B is blue-safe). We will see how to build and query the following data structure:

► **Definition 16.** A cross-safety tree T_S on S is a recursive partitioning on the plane similar to a kd-tree where we stop subdividing the points once we have reached a cross-safe subset of points. The root of T_S corresponds to S . If S is not cross-safe, we split the points around a vertical line L such that the two open halfplanes defined by L partition S into two sets S_1 and S_2 of size at most $\lceil |S|/2 \rceil$. The children of the root will then correspond to S_1 and S_2 . For every newly created node we repeat the procedure, partitioning the set of points by median x coordinates at even levels of the tree and median y coordinates at odd levels, until the points contained are a cross-safe subset of S .

The cell of a point p , denoted as C_p is the subset of points contained in the same leaf as p . A cell of T_S is red (resp. blue) if the points it contains are red (resp. blue).

The box of a point p , denoted as B_p is the smallest axis-aligned box containing all points in C_p (slightly extended to enforce our assumption of only considering boxes for which there are no points on the boundary of their four quadrants). A box of T_S is red (resp. blue) if the points it contains are red (resp. blue).

The box-point of p is the top-right point of the box of p , and has the same color as p .

A point is relevant if it has the minimum x or y coordinate among all points in its box (or equivalently, in its cell).

(See Figure 4 for an illustration of these definitions.)

Each node u in the tree stores:

- The set of point it contains, which we denote as P_u .

- The smallest axis-aligned bounding box of P_u , which we denote as B_u .
- The red box-points of minimum x and y coordinates among all box-points associated with a red point in P_u .
- The subset of all relevant blue points in the minimal set of P_u , sorted by x -coordinate.

Note that if the points in a minimal set are sorted by x coordinate then they are also sorted (in reverse order) by y coordinate.

4.3 Querying a cross-safety tree

Before we see how to build a cross-safety tree on S efficiently, let us see how we make queries on a node u of T_S . A query consists of a lower range $R_L = [x_L, +\infty] \times [y_L, +\infty]$ and an upper range $R_U = [-\infty, x_U] \times [-\infty, y_U]$ such that $R_L \cap R_U \neq \emptyset$ and neither the boundaries of R_L nor R_U intersect any blue box of T_S . It returns:

- rx_u , the minimum x -coordinate of any red box-points associated with a red point in $P_u \cap R_L$ (set to $+\infty$ if there are no red points in $P_u \cap R_L$).
- ry_u , the minimum y -coordinate of any red box-points associated with a red point in $P_u \cap R_L$ (set to $+\infty$ if there are no red points in $P_u \cap R_L$).
- bx_u , the minimum x -coordinate of any blue points in the minimal set of $P_u \cap R_L \cap R_U$ (set to $+\infty$ if there are no blue points in the minimal set of $P_u \cap R_L \cap R_U$).
- by_u , the minimum y -coordinate of any blue points in the minimal set of $P_u \cap R_L \cap R_U$ (set to $+\infty$ if there are no blue points in the minimal set of $P_u \cap R_L \cap R_U$).

Observe the following:

► **Observation 17.** *If a horizontal or vertical line passes through a red point then it does not intersect any blue-cross-safe box. The same applies when “red” and “blue” are swapped.*

► **Observation 18.** *Let B be a red-cross-safe box. Then all points in B dominate (resp. are dominated by) the same subset of blue points in S . The same applies when “red” and “blue” are swapped.*

We will need a few additional lemmas.

► **Lemma 19.** *The points corresponding to bx_u and by_u are relevant points of T_S .*

Proof. Let $P = P_u \cap R_L \cap R_U$, and suppose there is a blue point on the minimal set of P . Let p be the leftmost point in P which does not dominate any red point in P . Suppose that bx_u is not equal to $x(p)$. The only way this can happen is if p is not on the minimal set of P , meaning that there is a blue point q which is dominated by p (as p does not dominate any red point). In particular, q lies to the left of p . By definition of p , q thus dominates a red point. But if q dominates a red point and p dominates q , then p dominates a red point, which contradicts the definition of p . Thus $bx_u = x(p)$. Moreover, if p does not dominate any red point, then no point in its box dominates a red point, as the box is blue-cross-safe. Because the boundaries of R_L and R_U do not intersect any blue box, the whole box of p is contained in P . Thus, by definition of p , it is the leftmost point in its box and it is relevant. The same reasoning shows that by_u is the y coordinate of a relevant point. ◀

► **Lemma 20.** *If the bounding box B_u of points in P_u lies entirely in R_L , then we can return the necessary information in $O(\log n)$ time.*

Proof. In this case, rx_u and ry_u are already stored in the node, so we can return them in constant time. By Lemma 19, the point giving bx_u is the relevant blue point with minimum x coordinate among all blue points in the minimal set of $P_u \cap R_U$. This is also the relevant blue point with maximum y coordinate among all blue points in the minimal set of $P_u \cap R_U$.

We can do a binary search through the relevant blue points in the minimal set of P_u to find the point p below the line $y = y_U$ with maximum y coordinate. If this point is not in R_U (because $x(p) > x_U$) then no relevant blue point in the minimal set of P_u is in R_U , as all other relevant blue points q in the minimal set of P_u with $y(q) \leq y_U$ will have $x(q) > x(p) > x_U$. In this case we set $bx_u = +\infty$. Otherwise $bx_u = x(p)$.

We can find by_u similarly with a single binary search. \blacktriangleleft

► **Lemma 21.** *If u is not a leaf of T_S and B_u intersects the boundary of R_L , then we can return the necessary information after querying the children of u with the same lower range R_L (but possibly different upper ranges) and a constant amount of additional work.*

Proof. Suppose without loss of generality that the children of u split P_u by a vertical line (the situation for a horizontal line is similar). Let v be the child corresponding to the left half-plane and w the one corresponding to the right half-plane. Let us focus on computing bx_u , as this is the one requiring the most care.

Querying v with the same R_L and R_U returns some values rx_v , ry_v , bx_v and by_v . A blue point p on the minimal set of $P_w \cap R_L \cap R_U$ is a blue point on the minimal set of $P_u \cap R_L \cap R_U$ if and only if $y(p) \leq by_v$ and p does not dominate any red point in $P_v \cap R_U$. By Observation 18 this is equivalent to saying that $y(p) \leq by_v$ and $y(p) \leq ry_v$.

Thus we can compute bx_u by setting $R'_U = [-\infty, x_U] \times [-\infty, \min\{y_U, ry_v\}]$, querying w with R_L and R'_U to get values rx_w , ry_w , bx_w and by_w , then setting $bx_u = \min\{bx_v, bx_w\}$. Notice that by Observation 17, we are allowed to query w with R'_U as its boundary does not intersect any blue box of T_S . It is then easy to see that $rx_u = \min\{rx_v, rx_w\}$, $ry_u = \min\{ry_v, ry_w\}$ and $by_u = \min\{by_v, by_w\}$. \blacktriangleleft

► **Lemma 22.** *If u is a leaf of T_S and B_u intersects the lower boundary or the left boundary of R_L but not both simultaneously, then we can return the necessary information in constant time.*

Proof. In this case, we know that B_u is a red box, as by assumption the boundary of R_L does not intersect any blue box of T_S . Thus B_u contains no blue points and we know $bx_u = by_u = +\infty$. Suppose without loss of generality that B_u intersects the left boundary of R_L . Then the rightmost point of B_u is in R_L , and thus $P_u \cap R_L$ is non-empty. Because all points in P_u have the same box-point $p = (p_x, p_y)$, we have $rx_u = p_x$ and $ry_u = p_y$. \blacktriangleleft

► **Lemma 23.** *If u is a leaf of T_S and B_u intersects both the lower boundary and the left boundary of R_L , then we can return the necessary information after a single orthogonal range-emptiness test.*

Proof. Again, we know that B_u is a red box and all points in P_u have the same box point $p = (p_x, p_y)$. To know if we need to set $rx_u = ry_u = +\infty$ or $rx_u = p_x$ and $ry_u = p_y$, we simply need to test if there is a (necessarily red) point in $B_u \cap R_L$. This requires a single range-emptiness test. \blacktriangleleft

By applying the relevant result among Lemmas 20, 21, 22 and 23 recursively we get:

► **Theorem 24.** *We can query the root of a cross-safety tree T_S in $O(\sqrt{n} \log n)$ time plus the cost of a single range-emptiness test.*

As a corollary, we get:

► **Corollary 25.** *Let B be an axis aligned box. We can query a cross-safety tree T_S to test if B is red-NE-safe for S in $O(\sqrt{n} \log n)$ time plus the cost of a constant number of orthogonal range-emptiness tests on S .*

Proof. With two orthogonal range-emptiness queries on the blue points of S and one on the red points of S we can test if B contains at least one red point and $\text{cross}(B)$ contains only red points (that is, test if B is red-cross-safe for S). If B is not red-cross-safe for S we can immediately return “No”. We assume from now on that it is.

Let R_L be the range corresponding to $\text{NE}(B)$, and let $R_U = [-\infty, +\infty] \times [-\infty, +\infty]$. Because R is red-cross-safe, it is easy to see that the boundary of R_L does not intersect any blue box of T_S (this is also trivially true for R_U). Thus, we can query the root u of T_S with R_L and R_U to get the x coordinate bx_u of the blue point with minimum x coordinate among all blue points on the minimal set of $\text{NE}(B) \cap S$, or $+\infty$ if no such point exists. In particular, this allows us to test if such a point exists. ◀

We also have the following:

► **Lemma 26.** *Let $S' \subset S$ be a subset which conforms with S and let B be an axis-aligned box. If $S' \cap B \neq \emptyset$, then we can replace all orthogonal range-emptiness tests on S with the same tests on S' in the procedure described in Corollary 25 (including the tests done while querying T_S) without affecting the outcome.*

Proof. If there are both red and blue points in $S \cap \text{cross}(B)$, then at least one of these blue points participates in S . Because S' conforms with S , this blue point is also in S' , so S' is not red-cross-safe. The converse is trivially true. Thus, the initial three range-emptiness tests return the same results on S and S' .

Now consider the query done on T_S . If the corner of the lower range R_L does not intersect a red leaf box of T_S , then no range-emptiness test is performed and the claim holds. Now suppose R_L intersects a red leaf box B of T_S . Let S'' be the set of points in S where we remove all points in $S \cap B \cap R_L$ which are not in $S' \cap B \cap R_L$. Notice that by replacing the range-emptiness query on S with one on S' , the procedure behave exactly like querying a cross-safety tree on S'' . Because $S' \subset S'' \subset S$ we know that S'' conforms with S and thus by Observation 12 the claim holds. ◀

Thus, this data-structure fits the prerequisites of Theorem 15, and we can use it to get an algorithm solving the problem in $O(n(\mathcal{H}(S) + 1))$ time after having built it. The only missing ingredient to get an instance-optimal algorithm is building the data-structure within the same asymptotic runtime. We show in the following section that we can indeed do this.

4.4 Construction in $O(n(\mathcal{H}(S) + 1))$ time

Rather than focusing on constructing the cross-safety tree specifically, we start with a bit more general setting.

► **Theorem 27.** *Let S be a set of points. Let C be a property on axis-aligned bounding boxes of the plane such that for boxes $B_2 \subset B_1$, if $C(B_1)$ is true then $C(B_2)$ is true. (Note that C can depend on S).*

A partition Π of S is C -respectful if every set in Π is a singleton or can be enclosed by an axis aligned bounding box B such that $C(B)$ is true.

Let $\mathcal{H}_C(S)$ be the minimum of $\mathcal{H}(\Pi)$ over all C -respectful partitions of S .

If property $C(B)$ can be tested in $O(|S \cap B|)$ time when given access to $S \cap B$, then a kd-tree T_S^C with stopping condition C on the leaves can be built in $O(n(\mathcal{H}_C(S) + 1))$ time.

This remains true if for each node in the tree we do an additional linear amount of work (in the number of points considered at that node).

The proof is similar in spirit to that of Theorem 15, although simpler.

Proof. Consider the classical top-down recursive approach to construct a kd-tree on S (with linear-time median, selection), where we stop subdividing points once we have reached a bounding-box B with property C . Consider any C -respectful partition Π of S . Let $S_k \in \Pi$ and let B_k be a corresponding bounding box with the property C . At the j 'th level of the recursion, we have partitioned the plane into $O(2^j)$ boxes each containing at most $\lceil n/2^j \rceil$ points still to be considered. Any box B of T_S^C which is entirely contained in B_k has property C and can be set as a leaf. In other words the box B does not need to be recursed on and the points in $B \cap S$ are not considered at level j or lower. Because B_k intersects at most $O(\sqrt{2^j})$ boxes of T_S^C at level j , the number of points in S_k to consider at level j is $\min\{|S_k|, O(\sqrt{2^j} \cdot n/2^j)\} = \min\{|S_k|, O(n/\sqrt{2^j})\}$. At each level, the amount of work to be done is linear in the number of points to consider. The S_k 's cover the entire point set so by double summation we get that the runtime is in

$$\begin{aligned} O\left(\sum_j \sum_k \min\{|S_k|, n/\sqrt{2^j}\}\right) &= O\left(\sum_k \sum_j \min\{|S_k|, n/\sqrt{2^j}\}\right) \\ &= O\left(\sum_k \sum_{j \leq 2 \log(n/|S_k|)} |S_k| + \sum_{j > 2 \log(n/|S_k|)} n/\sqrt{2^j}\right) \\ &= O\left(\sum_k |S_k|(\log(n/|S_k|) + 1)\right) \\ &\subset O(n(\mathcal{H}_C(S) + 1)). \quad \blacktriangleleft \end{aligned}$$

Note that the proof generalizes easily to any constant dimension $d > 0$. We can apply this theorem to get the following (omitted proofs can be found in the full paper [12]):

► **Lemma 28.** *A set of points S can be preprocessed in $O(n(\mathcal{H}(S) + 1))$ time so that for any subset $S_k \subset S$, we can test if all points in S_k lie in a common vertical slab containing only points of S of the same color in $O(|S_k|)$ time.*

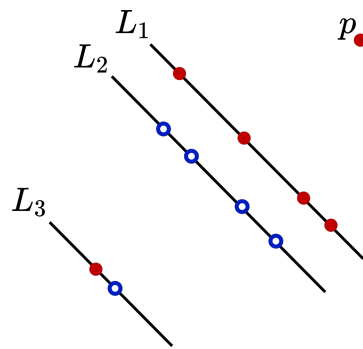
Which in turn implies:

► **Theorem 29.** *A cross-safety tree on S can be constructed in $O(n(\mathcal{H}(S) + 1))$.*

Finally, putting this together with 26 and Theorem 15 we get the main result.

► **Theorem 30.** *All points participating in S can be reported in $O(n(\mathcal{H}(S) + 1))$ time. In other words, there is an instance-optimal algorithm for this problem in the order-oblivious comparison-based model.*

One thing to note here is that while this guarantees that the algorithm is optimal with respect to any parameter of the instance which does not depend on the order of the input points, it is not immediately obvious that it runs in $O(n \log h)$ time, where h is the number of points to report (we only know that if there is an algorithm in the comparison-based model running within this time bound, then so does ours). The following results shows that the runtime of our algorithm is indeed within this bound.



■ **Figure 5** Type of instance considered in the impossibility proof.

► **Theorem 31.** *Let S be an instance of the Reporting participating points problem and let h be the number of points which participate in S . Then $n(\mathcal{H}(S) + 1) \in O(n \log h)$.*

5 Instance-optimality is impossible with linear queries

In the previous section, we have shown that there is a comparison-based algorithm to report participating points which is instance-optimal in the order-oblivious runtime against all comparison-based algorithm solving the problem. We also show that if we “compete” against algorithms which can do queries of the form $x(p) - x(q) \geq y(p) - y(q)$, then such a result is no longer possible, even if we allow our algorithm to be in a much stronger model of computation, such as algebraic computation trees. The full proof, in which we consider instances of the type illustrated in Figure 5, can be found in the full paper [12]. One caveat of this proof is that it relies on special instances with linear degeneracies (three points can be collinear). It is not clear if instance-optimality is possible when restricted to non-degenerate instances.

References

- 1 Peyman Afshani, J  r  my Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *J. ACM*, 64(1), 2017. doi:10.1145/3046673.
- 2 Pankaj K. Agarwal and Jir   Matousek. Relative neighborhood graphs in three dimensions. *Comput. Geom.*, 2:1–14, 1992. doi:10.1016/0925-7721(92)90017-M.
- 3 David Bremner, Erik D. Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, and Godfried T. Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. *Discret. Comput. Geom.*, 33(4):593–604, 2005. doi:10.1007/s00454-004-1152-0.
- 4 Jean Cardinal, S  bastien Collette, and Stefan Langerman. Empty region graphs. *Comput. Geom.*, 42(3):183–195, 2009. doi:10.1016/j.comgeo.2008.09.003.
- 5 Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discret. Comput. Geom.*, 16(4):361–368, 1996. doi:10.1007/BF02712873.
- 6 Timothy M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discret. Comput. Geom.*, 16(4):369–387, 1996. doi:10.1007/BF02712874.
- 7 Xiaomin Chen, J  nos Pach, Mario Szegedy, and G  bor Tardos. Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. *Random Struct. Algorithms*, 34(1):11–23, 2009. doi:10.1002/rsa.20246.
- 8 Olivier Devillers, Jeff Erickson, and Xavier Goaoc. Empty-ellipse graphs. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*,

- SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1249–1257. SIAM, 2008. URL: <http://dl.acm.org/citation.cfm?id=1347082.1347218>.
- 9 Ralf Hartmut Güting, Otto Nurmi, and Thomas Ottmann. Fast algorithms for direct enclosures and direct dominances. *J. Algorithms*, 10(2):170–186, 1989. doi:10.1016/0196-6774(89)90011-4.
 - 10 Manabu Ichino and Jack Sklansky. The relative neighborhood graph for mixed feature variables. *Pattern Recognit.*, 18(2):161–167, 1985. doi:10.1016/0031-3203(85)90040-8.
 - 11 J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992. doi:10.1109/5.163414.
 - 12 Justin Dallant Jean Cardinal and John Iacono. An instance-optimal algorithm for bichromatic rectangular visibility. *arXiv preprint arXiv:2106.05638*, 2021. arXiv:2106.05638.
 - 13 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discret. Comput. Geom.*, 7:13–28, 1992. doi:10.1007/BF02187821.
 - 14 David G. Kirkpatrick and John D. Radke. A framework for computational morphology. In Godfried T. TOUSSAINT, editor, *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 217–248. North-Holland, 1985. doi:10.1016/B978-0-444-87806-9.50013-X.
 - 15 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986. doi:10.1137/0215021.
 - 16 J. Ian Munro, Mark H. Overmars, and Derick Wood. Variations on visibility. In D. Soule, editor, *Proceedings of the Third Annual Symposium on Computational Geometry, Waterloo, Ontario, Canada, June 8-10, 1987*, pages 291–299. ACM, 1987. doi:10.1145/41958.41989.
 - 17 Mark H. Overmars. Connectability problems. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, volume 318 of *Lecture Notes in Computer Science*, pages 105–112. Springer, 1988. doi:10.1007/3-540-19487-8_11.
 - 18 Mark H. Overmars and Derick Wood. On rectangular visibility. *J. Algorithms*, 9(3):372–390, 1988. doi:10.1016/0196-6774(88)90028-4.
 - 19 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982. doi:10.1137/0211059.

Worst-Case Efficient Dynamic Geometric Independent Set

Jean Cardinal  

Université libre de Bruxelles (ULB), Brussels, Belgium

John Iacono  

Université libre de Bruxelles (ULB), Brussels, Belgium

Grigorios Koumoutsos 

Université libre de Bruxelles (ULB), Brussels, Belgium

Abstract

We consider the problem of maintaining an approximate maximum independent set of geometric objects under insertions and deletions. We present a data structure that maintains a constant-factor approximate maximum independent set for broad classes of fat objects in d dimensions, where d is assumed to be a constant, in sublinear *worst-case* update time. This gives the first results for dynamic independent set in a wide variety of geometric settings, such as disks, fat polygons, and their high-dimensional equivalents. For axis-aligned squares and hypercubes, our result improves upon all (recently announced) previous works. We obtain, in particular, a dynamic $(4 + \epsilon)$ -approximation for squares, with $O(\log^4 n)$ worst-case update time.

Our result is obtained via a two-level approach. First, we develop a dynamic data structure which stores all objects and provides an approximate independent set when queried, with output-sensitive running time. We show that via standard methods such a structure can be used to obtain a dynamic algorithm with *amortized* update time bounds. Then, to obtain worst-case update time algorithms, we develop a generic deamortization scheme that with each insertion/deletion keeps (i) the update time bounded and (ii) the number of changes in the independent set constant. We show that such a scheme is applicable to fat objects by showing an appropriate generalization of a separator theorem.

Interestingly, we show that our deamortization scheme is also necessary in order to obtain worst-case update bounds: If for a class of objects our scheme is not applicable, then no constant-factor approximation with sublinear worst-case update time is possible. We show that such a lower bound applies even for seemingly simple classes of geometric objects including axis-aligned rectangles in the plane.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Maximum independent set, deamortization, approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.25

Related Version *Full Version*: <https://arxiv.org/abs/2108.08050>

Funding *John Iacono*: Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.

Grigorios Koumoutsos: Supported by the Fonds de la Recherche Scientifique-FNRS under a “Scientific Collaborator” grant.

1 Introduction

Dynamic algorithms for classic NP-hard optimization problems is a quite active research area and major progress has been achieved over the last years (see [1, 9–11, 27]). For such problems, typically improved performance can be achieved when their *geometric* versions are considered, i.e., when the input has a certain geometric structure. As a result, a very recent line of research considers dynamic algorithms for well-known algorithmic problems in the



© Jean Cardinal, John Iacono, and Grigorios Koumoutsos;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 25; pp. 25:1–25:15

Leibniz International Proceedings in Informatics



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

geometric setting [3, 12, 18, 21, 29]. In this work, we continue this investigation by considering the geometric version of the Maximum Independent Set problem; some of our results improve upon the previous (very recent) work of Henzinger et al. [29] and Bhore et al. [12] and some others provide the first dynamic data structures for various geometric instances.

Static Independent Set

In the maximum independent set problem, we are given a graph $G = (V, E)$ and we aim to produce a subset $I \subseteq V$ of maximum cardinality, such that no two vertices in I are adjacent. This is one of the most well-studied algorithmic problems. It is well-known to be NP-complete and hard to approximate: no polynomial time algorithm can achieve an approximation factor $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless $P=NP$ [28, 36].

(Static) Geometric Independent Set

In the geometric version of the maximum independent set problem, the graph G is the intersection graph of a set V of geometric objects: each vertex corresponds to an object, and there is an edge between two vertices if and only if the corresponding objects intersect.

Besides the theoretical interest, the study of independent sets of geometric objects such as axis-aligned squares, rectangles or disks has been motivated by applications in various areas such as VLSI design [30], map labeling [4, 35] and data mining [8, 31].

For most classes of geometric objects, the problem remains NP-hard. A notable exception is the case where all objects are intervals on the real line (the well-known *interval scheduling* problem), where the optimal solution can be computed in polynomial time using a folklore greedy algorithm [25]. However, even the simplest 2d-generalization to axis-aligned unit squares is NP-hard [24]. As a result, most of the related work has focused on developing PTASs for certain versions of geometric independent set. This has been achieved for various types of objects such as squares, hypercubes, fat objects and pseudodisks [15, 16, 23, 30].

A substantially harder setting is the when the geometric objects are arbitrary (axis-aligned) rectangles in the plane: Despite the intense interest (see [2, 20]), no PTAS is known. Until very recently the best known approximation factor was $O(\log \log n)$ [13, 14]; in 2021, a $O(1)$ -approximation was claimed [33].

Dynamic Geometric Independent Set

In the dynamic version of geometric independent set, V is a class of geometric objects (for instance squares in the plane) and we maintain a set $S \subseteq V$ of *active* objects. Objects of V may be inserted in S and deleted from S over time and the goal is to maintain a maximum independent set¹ of S , while keeping the time to implement the changes (the *update time*) sublinear on the size of the input.

Previous Work

Most previous work dates from 2020 onwards. The only exception is the work of Gavruskin et al. [26] who considered the very special case of intervals where no interval is fully contained in any other interval. The study of this area was revitalized by a paper of Henzinger,

¹ This problem should not be confused by the related, but very different, *maximal* independent set problem, where the goal is to maintain an inclusionwise maximal independent set subject to updates on the edges (and not the vertices). This problem has been studied in the dynamic setting, with a remarkable recent progress after a sequence of breakthrough results [5–7, 19].

Neumann and Wiese in SoCG'20 [29]. They presented deterministic dynamic algorithms with approximation ratio $(1 + \epsilon)$ for intervals and $(1 + \epsilon) \cdot 2^d$ for d -dimensional hypercubes, under the assumption that all objects are contained in $[0, N]^d$ and each of their edges has at least unit length. Their update time was polylogarithmic in both n and N , but N could be unbounded in n . Furthermore, they showed that no $(1 + \epsilon)$ -approximation scheme with sublinear update time is possible, unless the Exponential Time Hypothesis fails.

Subsequently Bhore et al. [12] obtained the first results for dynamic geometric independent set without any assumption on the input. For intervals, they presented a dynamic $(1 + \epsilon)$ -approximation with logarithmic update time; this was recently simplified and improved by Compton et al. [21]. For squares, [12] presented a randomized algorithm with expected approximation ratio roughly 2^{12} (generalizing to roughly 2^{2d+5} for d -dimensional hypercubes) with amortized update time $O(\log^5 n)$ (generalizing to $O(\log^{2d+1})$ for hypercubes).

Our Results

In this work we obtain the first dynamic algorithms for fat objects (formally defined in Section 2) in fixed dimension d with sublinear worst-case update time. The precise bounds on the approximation ratio and the update time depend on the fatness constant and the running time for worst-case range query structures for each family of objects. However, the results follow from a generic approach, applicable to all those classes of objects.

► **Theorem 1.** *There exists a deterministic data structure for maintaining a $O(1)$ -approximate independent set of a collection of fat objects, with sublinear worst-case update time, that reports $O(1)$ changes in the independent set per update. In particular, it achieves the following approximation ratios and worst-case update times, for any constant $0 < \epsilon \leq 1/4$:*

- $(4 + \epsilon)$ -approximation with $O(\log^4 n)$ update time for axis-aligned squares in the plane.
- $(5 + \epsilon)$ -approximation with $O(n^{3/4})$ update time for disks in the plane.
- $(2^d + \epsilon)$ -approximation with $O(\log^{2d} n)$ update time for d -dimensional hypercubes.
- $O(1)$ -approximation with $O(\log^{2dk} n)$ update time for fat objects which are unions of k (hyper)rectangles
- $O(1)$ -approximation with $O(n^{1 - \frac{1}{d(d+1)}})$ update time for fat simplices in d dimensions.
- $O(1)$ -approximation with $O(n^{1 - \frac{1}{d(d+1)k}})$ update time for fat objects which are unions of k simplices in d dimensions.
- $O(1)$ -approximation with $O(n^{1 - \frac{1}{d+2}})$ update time for balls in d dimensions.

This result gives the first dynamic algorithms with sublinear update time for all the aforementioned classes of objects, apart from d -dimensional hypercubes. Moreover, for hypercubes our result improves upon both of [29] and [12]. In [29] the approximation ratio is the same as ours, but here we achieve this with no extra assumption on the input and our running time $O(\log^{2d} n)$ is faster than the $O(\log^{2d+1} n \cdot \log^{2d+1} N)$, especially since N might be arbitrarily large. Compared to [12], our result is stronger with respect to all three metrics: the approximation factor, the update time, and the worst-case behavior.

In fact, it seems hard to significantly improve our result on any aspect: First, for the approximation factor, as mentioned above, Henzinger, Neumann and Wiese [29] proved that (under the Exponential Time Hypothesis) one cannot maintain a $(1 + \epsilon)$ -approximate maximum independent set of hypercubes in $d \geq 2$ dimensions with update time $n^{O((1/\epsilon)^{1-\delta})}$ for any $\delta > 0$. Moreover, the update time we obtain is essentially the time required for detecting intersections between objects in a range query data structure. Fatness of the objects is a sensible condition for achieving such results: we prove that for (nonfat) rectangles,

ellipses, or arbitrary polygons, no dynamic approximation in sublinear worst-case update time is possible. Finally, we emphasize the remarkable additional property that the number of changes reported per update is always constant.

To obtain the result of Theorem 1 we develop a generic method to obtain dynamic independent set algorithms and show how to apply it for fat objects. Our method uses a combination of several components. The first is a data structure which stores all objects, supports insertions and deletions in sublinear $f(n)$ time and, when queried, returns a β -approximate independent set with output-sensitive running time. The second main component is a generic transformation of such a data structure into a dynamic algorithm with approximation factor $\beta + \epsilon$ and *amortized* update time $O(f(n))$. This is done by periodically updating the solution using the data structure. Then, we apply a generic *deamortization* scheme, involving what we call a *MIX* function, a generic way to switch “smoothly” from one solution to another. We design such a MIX function for fat objects using geometric separators.

2 Overview

Notation

In what follows, V is a class of geometric objects, such as squares in the plane, and we consider a finite set $S \subseteq V$. A subset of S is a *maximum independent set* (MIS) of S if all its objects are pairwise disjoint and it has maximum size over all sets with this property. We use OPT to denote the size of a maximum independent set and let $n = |S|$. We say that I is a β -approximate MIS if its size is at least βOPT (here we adopt the convention that β is a constant $0 < \beta \leq 1$).

The problem we study is to maintain an approximate MIS set I under a sequence of insertions and deletions in S , which we call generically an *update*, starting from an empty set. This can be expressed as implementing a single operation $\Delta = \text{UPDATE}(u)$, where u is the object to be inserted or deleted, and Δ , the *update set*, is the set of objects that change in the approximate MIS I , presented as the symmetric difference from the previous set. In general we will use subscripts to express variables’ states after the indicated update, and unsubscripted versions for the current state. Using the operator \oplus to denote the symmetric difference, we therefore have $S_i := \oplus_{j=1}^i \{u_j\}$ and $I_i := \oplus_{j=1}^i \Delta_j$. We say that UPDATE is a β -approximate MIS algorithm if I_i is always a β -approximate MIS of S_i . We adopt the convention that the update set Δ_i is always returned explicitly and thus the running time of an update u_i is at least the size of the returned update set, $|\Delta_i|$.

We begin with a simple yet crucial observation about MIS.

► **Fact 2.** *The size of a MIS can change by at most one with every update: $|\text{OPT}_i - \text{OPT}_{i-1}| \leq 1$.*

Note that this fact does not hold for the weighted version of the MIS problem. Also note that this does not say that it is possible to have an update algorithm with an update set Δ_i with cardinality always at most 1; this fact bounds how the size of a MIS can change after an update, and not the content. In fact, it is easy to produce examples where $|\Delta_i|$ must be 2, even for intervals (e.g. $u_1 = [1, 4]$, $u_2 = [2, 3]$, $u_3 = [1, 4]$). However, it does leave open the possibility to have an update operation returning constant-size update sets, which is something we will achieve for the classes of geometric objects we consider.

Overview

Our goal is thus to develop a general method for dynamic approximate MIS that has efficient worst-case running times and small update set sizes for various classes of geometric objects. We identify two ingredients that are needed:

β -dynamic independent set query structure (β -DISQS): This is an abstract data type for maintaining a set S of n objects, in which one can insert or delete an object in time $f(n)$, and obtain a β -approximate MIS of the current set S in output-sensitive time $kf(n)$ if the set returned has size k .

MIX algorithm: A MIX algorithm receives two independent sets S_1 and S_2 whose sizes sum to n as input, and smoothly transitions from S_1 to S_2 by adding or removing one element at a time such that at all times the intermediate sets are an independent sets of size at least $\min(|S_1|, |S_2|) - o(n)$. The running time of the MIX algorithm is said to be $\gamma(n)$ if the entire computation takes time $n\gamma(n)$.

The plan of the paper is as follows.

Section 3: Amortized Update Time

In this section, we prove that the first ingredient, the β -DISQS, is sufficient to obtain a $(\beta - \epsilon)$ -approximate dynamic MIS algorithm with $O(f(n))$ amortized update time for any $\epsilon > 0$. This is presented as Theorem 4. We note that this is a bit of a “folklore” algorithm that essentially does nothing more than periodically call QUERY.

Section 4: Worst-case Update time

The second ingredient, MIX, is vital to deamortizing: In Section 4, we show that a DISQS and a MIX together are sufficient to produce a data structure which for any constant $\epsilon > 0$ maintains an approximate MIS of size at least $(\beta - \epsilon) \text{OPT} - o(\text{OPT})$, has a worst-case running time of $O(f(n) + \gamma(n) + \log n)$, and whose UPDATE operation returns a constant-sized update set; this is presented as Theorem 7. We also show in that the non-existence of a MIX function implies the impossibility of an approximate MIS data structure with sublinear update set size, hence sublinear worst-case running time, and that this impossibility holds for rectangles and other classes of nonfat objects (Theorem 8 and Lemma 9).

Given this generic scheme developed, to prove our main result, it remains to show that fat objects have a MIX function and a DISQS. This is the content of Sections 5 and 6.

Section 5: Existence of a MIX function for fat objects

We show in Lemma 13 that for fat objects in any constant dimension a MIX algorithm always exists with $\gamma(n) = O(\log n)$. This is achieved via geometric separators [34].

Section 6: Existence of β -DISQS for fat objects

Last, we show that obtaining a β -DISQS is possible for many types of objects using variants of standard range query results, with the running time $f(n)$ matching what we expect for range queries: polylogarithmic for orthogonal objects of constant complexity, and $O(n^c)$, $c < 1$ for non-orthogonal objects such as disks, triangles or general polyhedra. The result is based on a simple greedy algorithm for the MIS problem on fat objects [22, 32], yielding an approximation factor β that only depends on the fatness constant.

3 Dynamization

► **Definition 3.** A β -dynamic independent set query structure (β -DISQS) is a data structure that maintains a set S whose size is denoted as n and supports the following operations:

- **UPDATE(u):** Insert or remove u , so that S becomes $S \oplus u$.
- **QUERY:** Returns a β -approximate MIS of S

We say that the running time of a DISQS is $f(n)$ if UPDATE takes time $f(n)$ and if QUERY returns a set of size k in time $kf(n)$. We require $f(n)$ to be sublinear.

We now show that a β -DISQS is sufficient to give a approximate MIS data structure with an amortized running time, with a loss of only ϵ in the approximation factor for any $\epsilon > 0$. The intuition is simple, pass through the update operations to the DISQS and periodically replace the approximate MIS seen by the user by querying the DISQS. The only subtlety is to immediately remove any items from the approximate MIS that have been deleted in order to keep the approximate MIS valid. This simple transformation is likely folklore, but we work out the details in the full version of this paper for completeness.

► **Theorem 4.** Given a β -DISQS with sublinear running time $f(n)$ for an independent set problem and a $\epsilon > 0$ there is a fully dynamic data structure to maintain a $(1-\epsilon) \cdot \beta$ -approximate independent set that runs in $O(\frac{1}{\epsilon}f(n) + \log n)$ amortized time per operation.

4 Deamortization

In this section we present our deamortization technique. In particular, we describe a procedure, which we call MIX, which if exists, is used to transform a β -DISQS into a deterministic dynamic algorithm for with worst-case update time guarantees and update set size bounds. We also show that if a MIX does not exist for an independent set problem, then no sublinear worst-case update time guarantees are possible.

MIX function

We now define our main ingredient for deamortization, which essentially says that we can smoothly switch from one solution to another, by adding or removing one item at a time:

► **Definition 5 (MIX function).** Given two solution sets A and B , let $\text{MIX}(A, B, i)$, for $i \in [0, |A| + |B|]$ be a set where:

- $\text{MIX}(A, B, i)$ is always a valid solution
- $\text{MIX}(A, B, 0) = A$
- $\text{MIX}(A, B, |A| + |B|) = B$
- $|\text{MIX}(A, B, i)| \geq \min(|A|, |B|) - \Gamma(|A| + |B|)$, for some $\Gamma(|A| + |B|) = o(|A| + |B|)$.
- $\text{MIX}(A, B, i)$ and $\text{MIX}(A, B, i + 1)$ differ by one item.

Given this purely combinatorial definition, we define a MIX algorithm as follows.

► **Definition 6.** A MIX algorithm with running time $\gamma(n)$ is a data structure such that

1. It is initialized with $A, B, i = 0$ and it has a single operation ADVANCE which advances i and reports the single element in the symmetric difference $\text{MIX}(A, B, i) \oplus \text{MIX}(A, B, i - 1)$.
2. The initialization plus $|A| + |B|$ calls to ADVANCE run in total time $(|A| + |B|) \cdot \gamma(|A| + |B|)$.

The rest of this section is organized as follows. In 4.1 we show that given a β -DISQS and a MIX function for an independent set problem, we can produce a dynamic algorithm with worst-case update time guarantees and approximation ratio arbitrarily close to β . In

4.2, we show the necessity of MIX function; in other words, we show that if there does not exist a MIX function for an independent set problem, then no deterministic algorithms with worst-case update time guarantees exist.

4.1 Dynamic algorithm with worst-Case update time

We now present our main theorem. As we discuss next, the intuition expands upon that for Theorem 4, that in addition to periodically using the β -DISQS to get a new solution, the MIX slowly transitions between the two previous solutions reported by the β -DISQS. Our result is the following.

► **Theorem 7.** *Given a β -DISQS with running time $f(n)$ for an independent set problem, a $\gamma(n)$ -time MIX algorithm with nondecreasing $\Gamma(n) = o(n)$, and an $0 < \epsilon < 1/4$, there is a fully dynamic data structure to maintain an independent set of size at least $(\beta - \epsilon) \text{OPT} - o(\text{OPT})$. The data structure runs in $O_{\epsilon, \beta}(f(n) + \gamma(n) + \log n)$ worst-case time per operation, where n is the current number of objects stored, and reports a $O_\epsilon(1)$ number of changes in the independent set per update.*

High-level description

We saw in the previous section how to obtain an amortized update time algorithm by splitting the update sequence into *rounds* and query the DISQS at the end of each round to recompute an approximate MIS. Let \hat{I}_k be the independent set produced by the DISQS at the end of round k . At a high-level, the main task is to deamortize the computation of \hat{I}_k : we can not afford computing it during one step. Instead, we compute \hat{I}_k gradually during round $k + 1$, making sure that the running time per step is bounded. \hat{I}_k is eventually computed by the end of round $k + 1$. At this point, we would like to have \hat{I}_k (discarding elements deleted during round $k + 1$) as our output independent set; however this can not be done immediately, since \hat{I}_k might be very different from \hat{I}_{k-1} . For that reason, the switch from \hat{I}_{k-1} to \hat{I}_k is done gradually using the MIX function during round $k + 2$. After all, our algorithm uses \hat{I}_k as its independent set at the end of round $k + 2$.

It follows that the independent set reported to the user is a combination of DISQS queries 3 or 2 rounds in the past. We show that by appropriately choosing the lengths of the rounds depending on the sizes of the independent sets, this lag affects the approximation factor by an additive ϵ .

Proof of Theorem 7. We group the updates u_i into rounds, and use r_k to indicate that u_{r_k} is the last update of round k . Let \hat{I}_k be the independent set output by the β -DISQS (if queried) at time r_k , i.e., at the end of k th round. The length of the k th round, $R_k = r_k - r_{k-1}$ is defined to be $R_k := \max\{1, \lfloor \epsilon \cdot |\hat{I}_{k-2}| \rfloor\}$ updates.

For convenience, we define the following functions for any $0 < \epsilon < 1/4$:

$$g(\epsilon) = \frac{1 + \sqrt{1 - 4\epsilon}}{2} \quad h(\epsilon) = \frac{3 - \sqrt{1 - 4\epsilon}}{2} \quad \phi(\epsilon) = \frac{16h^2(\epsilon)}{\epsilon \cdot \beta \cdot g^3(\epsilon)}$$

Note that $g(\epsilon) = 1 - \epsilon - O(\epsilon^2)$ and $h(\epsilon) = 1 + \epsilon + O(\epsilon^2)$. Note also that $g(\epsilon) \in (1/2, 1)$ and in particular $g(\epsilon) \rightarrow 1$ as $\epsilon \rightarrow 0$. Similarly, $h(\epsilon) \in (1, 3/2)$ and $h(\epsilon) \rightarrow 1$ as $\epsilon \rightarrow 0$.

Our Data Structures and Operations

Our overall structure contains a β -DISQS and a MIX algorithm. We also maintain the current active set of objects S_i and our approximate independent set I_i explicitly in a binary search tree; we refer to simply as S and I , each stored according to some total order on the objects². We maintain the invariant that at the beginning of round k , the DISQS stores the set of objects $S_{r_{k-2}}$. Also, our structure stores \hat{I}_{k-3} and \hat{I}_{k-2} .

To execute update operations in round k , the following are performed:

Running MIX slowly: During round k we use MIX to transition from \hat{I}_{k-3} to \hat{I}_{k-2} . This is done by initializing MIX with \hat{I}_{k-3} and \hat{I}_{k-2} and repeatedly running the ADVANCE operation. After each update, we continue running MIX where we left off and continue until either $\phi(\epsilon) \cdot \gamma(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|)$ time has passed or if $\phi(\epsilon)$ calls to ADVANCE have been performed.

Operation archiving: All updates are placed into a queue Q as they arrive. This will ensure that the DISQS will take into account all updates of previous rounds. Moreover, if an update u_i deletes an element v of S , we wish to report it as being deleted. To do so we set a variable $\Delta_i^{\text{DEL}} = \{v\}$, and otherwise $\Delta_i^{\text{DEL}} = \emptyset$.

Interaction with the DISQS: During round k , we want to use the DISQS in order to compute the set \hat{I}_{k-1} . To do that, we first perform to the DISQS all updates of round $k-1$ one by one and remove them from Q . This way, DISQS stores the set $S_{r_{k-1}}$. Then, we perform a query to the DISQS, to get \hat{I}_{k-1} . In each update of the round $\left(1 + \frac{2h(\epsilon)}{\beta\epsilon}\right) f(n)$ time is spent on executing these operations.

Maintaining S : We store S in a binary search tree based on some total ordering of the objects. For each update u_i , we search for u_i in S and remove it if it is there, and add it if it is not, to maintain $S = S_i = S_{i-1} \oplus \{u_i\}$.

Output: After each update, we report the symmetric difference Δ between previous and current independent set to the user. Let Δ_i^{MIX} be the union of the items returned by the ADVANCE operation of the MIX algorithm during the execution of update u_i . We combine Δ_i^{MIX} and Δ_i^{DEL} , and before returning, we remove any items that would result in the insertion into I_i of items that are not in S .

Roadmap

We need to argue about (i) correctness, (ii) running time, (iii) approximation ratio and (iv) *feasibility* of our algorithm, i.e., that during each round the computation of MIX and DISQS have finished before the round ends. Due to space limitations, we provide a brief sketch here and defer the formal proofs to the full version.

Correctness

It is easy to see that the algorithm described above always outputs an independent set to the user: During current round t , the user always sees $\text{MIX}(\hat{I}_{t-3}, \hat{I}_{t-2}, j)$ for some j , with perhaps some items that have been deleted in round $t-1$ or the current round removed. MIX is by definition an independent set at every step as in any subset of it, so the user always sees an independent set.

² Instead of a tree one could use a hash table, which would remove the additive logarithmic term from the update time, at the expense of randomization. However this will not improve our overall result, since functions $\gamma(n)$ and $f(n)$ are at least logarithmic in our application (see sections 5,6); therefore the logarithm can be absorbed while keeping the result deterministic.

Bound on update set size

The update set returned is a subset of $\Delta_i^{\text{MIX}} \cup \Delta_i^{\text{DEL}}$, as insertions of Δ_i^{MIX} might be removed if the items are no longer in S . The size of Δ_i^{MIX} is at most $\phi(\epsilon)$ by construction; the set Δ_i^{DEL} contains at most one element. Therefore,

$$\Delta \leq \phi(\epsilon) + 1 = \frac{16h^2(\epsilon)}{\epsilon \cdot \beta \cdot g^3(\epsilon)} + 1 \leq \frac{16}{\epsilon \cdot \beta} \cdot \frac{(3/2)^2}{(1/2)^3} + 1 = O\left(\frac{1}{\epsilon \cdot \beta}\right) = O_{\epsilon, \beta}(1) \quad (1)$$

Running time

The formal proof is located in the full version. Briefly, MIX takes time $\phi(\epsilon) \cdot \gamma(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|)$. It is easy to show that $|\hat{I}_{k-3}| + |\hat{I}_{k-2}| \leq \frac{2\text{OPT}}{g^3(\epsilon)} \leq 16n$. Using that γ is sublinear, we get that $\phi(\epsilon) \cdot \gamma(|\hat{I}_{k-3}| + |\hat{I}_{k-2}|) = O_{\epsilon}(\gamma(n))$. DISQS works for time $(1 + \frac{2h(\epsilon)}{\beta \cdot \epsilon})f(n) = O_{\epsilon, \beta}(f(n))$. Last, operations for checking if update elements are still in S require time $O(|\Delta_i^{\text{MIX}} \cup \Delta_i^{\text{DEL}}| \cdot \log n) = O_{\epsilon, \beta}(\log n)$, due to (1).

Approximation ratio and feasibility

The proofs of approximation ratio and feasibility are simple but technical, and deferred to the full version. At a high level, the main idea is that the size of each round is carefully chosen such that, combined with Fact 2, the approximation factor β of the DISQS worsens just by an additive ϵ term due to the delayed updates and allows the DISQS and MIX operation to complete during the round. ◀

4.2 Necessity of the MIX function

► **Theorem 8.** *Suppose for any n , there are independent sets A and B of size n such there is no MIX function with $\Gamma(n) \leq (1 - \beta)n$ for all $A' \subseteq A$ and $B' \subseteq B$, where $|A'|, |B'| \geq \beta n$. Then there is no $(\beta + \epsilon)$ -approximate dynamic MIS algorithm that reports at most $o(n)$ changes in the independent set per update, for any $\epsilon > 0$.*

Proof. Let $\delta(n) = o(n)$ and suppose there is a $(\beta - \epsilon)$ -approximate MIS algorithm that reports $\delta(n)$ changes per update in the worst case. Insert A in to the data structure. Then insert B and delete A . This is $|A| + |B|$ update operations. At all times the independent set is at least $(\beta - \epsilon)n$, and there are at most $\delta(n)$ changes per update operation. We could transform this into a MIX function by taking the at most $\delta(n)$ changes from update and report each change one at a time, first deletes and then inserts; thus at each step of the resultant MIX, their independent set is at least $(\beta + \epsilon)n - \delta(n)$ which is at least βn for sufficiently large n . ◀

We can apply this to the case of rectangles in the plane, where we show that with a non-trivial worst-case performance of $o(n)$ changes in the independent set per operation, it is impossible to have an β -approximate MIS for any $\beta > 0$.

► **Lemma 9.** *There is no MIX function for rectangles with $\Gamma(n) < n$. Thus from Theorem 8, for any $\beta > 0$, there is no β -approximate dynamic MIS algorithm that reports at most $o(n)$ changes in the independent set per update.*

Proof. This is equivalent to saying that there are sets of rectangles A and B such that for any MIX function, there is an i such that $\text{MIX}(A, B, i) = \emptyset$. Consider sets of rectangles A and B , each of size n , in the form of a grid such that A are horizontally thin and disjoint, B

are vertically thin and disjoint, and every rectangle of A intersects all rectangles of B . In a MIX function, starting from A , one can not add a single element from B until all elements of A have been removed. ◀

This construction works for any class of objects for such a generalized “hashtag” construction is possible, which includes any class of shapes which are connected and where for any rectangle, there is a shape in the class that has that rectangle as its minimum orthogonal bounding rectangle. This includes natural classes of shapes without fatness constraints, such as triangles, ellipses, polygons, etc.

5 A MIX algorithm for fat objects

Fat objects

There are many possible definitions of fat objects in Euclidean space, we use the following one from [15]. Define the center and size of an object to be the center and side length of one of its minimal enclosing hypercube.

► **Definition 10.** *A collection S of (connected) objects is f -fat, for a constant f , if for any size- r box R , there are f points such that every object that intersects R and has size at least r contains one of the chosen points.*

This implies that any box can only intersect f disjoint objects of size larger than the box. Throughout the whole section, f and the dimension d are considered to be constant.

We will develop and use a variant of the rectangle separator theorem of Smith and Wormald [34]. We first state the classic version, and then prove the variant we need. Our proofs are straightforward adaptations of those in [34].

► **Lemma 11** (Smith and Wormald [34]). *For any set S of disjoint squares objects in the plane, there is a separating rectangle R such that if we partition S into S^{IN} , S^{OUT} and S^{ON} based on whether each object lies entirely inside R , entirely outside R , or intersects R , $|S^{\text{IN}}| \leq \frac{2}{3}|S|$, $|S^{\text{OUT}}| \leq \frac{2}{3}|S|$ and $|S^{\text{ON}}| = O(\sqrt{|S|})$.*

What we need differs from this in that we have two sets of fat objects, in each set the objects are disjoint but intersection is allowed between the two sets, and we want to have the separator intersect with an order-square-root number of objects in each set. However we require the separator to be balanced with respect to the first set only; it is not possible to require balance with respect to both sets. We state the separator theorem here, the proof is in the full version.

► **Lemma 12.** *Let S_1 and S_2 be two sets of disjoint f -fat objects in d -dimensions. Let $n = |S_1| + |S_2|$. We can compute a hypercube s and sets $S_1^{\text{IN}}, S_2^{\text{IN}}, S_1^{\text{OUT}}, S_2^{\text{OUT}}$ with the following properties in time $O(d \cdot n) = O(n)$:*

- *The hypercube s intersects $O(n^{1-\frac{1}{d}})$ objects of $S_1 \cup S_2$.*
- *$S_1^{\text{IN}} \subseteq S_1$, $S_2^{\text{IN}} \subseteq S_2$, $S_1^{\text{OUT}} \subseteq S_1$, $S_2^{\text{OUT}} \subseteq S_2$*
- *All objects in S_1^{IN} and S_2^{IN} lie entirely inside s*
- *All objects in S_1^{OUT} and S_2^{OUT} lie entirely outside s*
- *$|S_1^{\text{IN}}| \leq \frac{4^d-1}{4^d}|S_1|$*
- *$|S_1^{\text{OUT}}| \leq \frac{4^d-1}{4^d}|S_1|$*

► **Lemma 13.** *Fat objects in constant dimension d have a MIX algorithm with running time $\gamma(n) = O(\log n)$: Given independent sets of fat objects S_1 and S_2 , there is a MIX from S_1 to S_2 whose size is always at least $\min(|S_1|, |S_2|) - \Gamma(|S_1| + |S_2|)$, with $\Gamma(n) = O(n^{1-\frac{1}{d}} \log n) = o(n)$. The total running time of initializing the algorithm with S_1 and S_2 and performing all steps of MIX is $O((|S_1| + |S_2|) \cdot \log(|S_1| + |S_2|))$.*

Proof. We compute the separator s from Lemma 12. Let $S_1^{\text{IN}}, S_1^{\text{OUT}}, S_1^s, S_2^{\text{IN}}, S_2^{\text{OUT}}, S_2^s$, denote partition of S_1 and S_2 into parts that are completely inside the separator, completely outside, and those that intersect the separator, respectively.

Main Idea. The main idea is the following: First, we will remove all objects of S_1^s . Then the remaining objects of S_1 would be either completely inside s or completely outside S . We will use recursively the MIX function in both sides to switch from S_1^{IN} to S_2^{IN} and from S_1^{OUT} to S_2^{OUT} respectively. At the end we will add S_2^s .

Applying the recursion carefully. Recall that our goal is twofold: First, minimize the running time of MIX, second, make sure that at each step the size of the current set is at large as possible. Formally, minimize both $\gamma(n)$ and $\Gamma(n)$, which should be definitely sublinear, and as small as possible. Note that towards the second goal, i.e., keeping the set size as high as possible at all times, the sets used to apply the first recursive call of MIX matter: should we start from switching from S_1^{IN} to S_2^{IN} or from S_1^{OUT} to S_2^{OUT} ? We want to make the choice that leads to the largest independent set at the intermediate step when only one of the two recursive calls has been applied.

We denote a and b the sides of separator (IN and OUT) so that $|S_1^a| + |S_2^b| \leq |S_1^b| + |S_2^a|$ holds. As $\min(w + x, y + z) \leq \max(w + z, x + y)$:

$$|S_1^b| + |S_2^a| = \max(|S_1^b| + |S_2^a|, |S_1^a| + |S_2^b|) \geq \min(|S_1^a| + |S_1^b|, |S_2^a| + |S_2^b|). \quad (2)$$

This way, at the intermediate step when we have mixed only one side, the set size will not be smaller than the beginning or the end of the mix operation.

Formal Description. The MIX function then proceeds as follows:

1. Start with S_1 .
2. Remove the elements of S_1^s , one at a time, to give $S_1^a \cup S_1^b$.
3. Recursively MIX S_1^a to S_2^a .
4. Now we have $S_2^a \cup S_1^b$.
5. Recursively MIX S_1^b to S_2^b . At the end of this process we will have $S_2^b \cup S_2^a$.
6. Add the elements of S_2^s , one at a time.
7. We finish with $S_2^a \cup S_2^b \cup S_2^s = S_2$.

The base case is when one of the two sets is empty, and the MIX proceeds in the obvious way by deleting all elements of S_1 if S_2 is empty, or inserting all elements of S_2 if S_1 is empty. In such a case the lemma holds trivially as $\min(|S_1|, |S_2|)$ is zero.

We need to argue that at all times this process generates a set that is an independent set of the claimed size.

Always an independent set. In steps 1-2 we always have a subset of S_1 , which is an independent set, the same holds in steps 6-7 with respect to S_2 . In step 4, $S_2^a \cup S_1^b$ is independent as each of the sets are independent and all of the objects on each set are entirely on opposite sides of the separating rectangle s . Steps 3 and 5 hold by induction, and that the part we are recursively MIXing and the part that is unchanged are entirely on opposite sides of the separating rectangle.

Size bound. Let $\text{MIX}_{\min}(S_1, S_2, n)$ be the smallest size of the independent set during the running of the MIX function from S_1 to S_2 , and where n is an upper bound on $|S_1| + |S_2|$. Then we can directly express MIX_{\min} as a recurrence, taking the minimum of each step of the algorithm:

$$\begin{aligned} \text{MIX}_{\min}(S_1, S_2, n) = \min(&|S_1|, |S_1^a| + |S_1^b|, \text{MIX}_{\min}(|S_1^a|, |S_2^a|, n) + |S_1^b|, |S_2^a| + |S_1^b|, \\ &|S_2^a| + \text{MIX}_{\min}(|S_1^b|, |S_2^b|, n), |S_2^a| + |S_2^b|, |S_2|) \end{aligned}$$

We will prove that

$$\text{MIX}(S_1, S_2, n) \geq \min(|S_1|, |S_2|) - (\log_{\frac{4^d}{4^d-1}} |S_1|) \cdot n^{1-\frac{1}{d}} \quad (3)$$

which implies the claim of this lemma. The details of this inductive proof are given in the full version. It follows exactly the intuition that (i) the only loss in the MIX function is the separators at each level of the recursion, (ii) there are logarithmic levels of the recursion and (iii) the size of the separators in each level are $O(n^{1-1/d})$. Some care must be taken, since the separators are only balanced for one of the two sets.

Running time. The running time is given by the recursion:

$$T(S_1, S_2) = \begin{cases} |S_1| & \text{if } S_2 = \emptyset \\ |S_2| & \text{if } S_1 = \emptyset \\ T(S_1^a, S_2^a) + T(S_1^b, S_2^b) + O(|S_1| + |S_2|) & \text{otherwise} \end{cases}$$

where the additive term in the last case is dominated by the time needed to compute the separator (Lemma 12) which is $O(d \cdot (|S_1| + |S_2|)) = O(|S_1| + |S_2|)$, since d is assumed to be constant.

Recall $|S_1^a|, |S_2^a| \leq \frac{4^d}{4^d-1} |S_1|$, S_1^a and S_1^b are disjoint subsets of S_1 , and S_2^a and S_2^b are disjoint subsets of S_2 . Hence the recursion depth is logarithmic in $|S_1|$ and each item from S_1 and S_2 is passed to at most one of the recursive terms.

Thus the running time is $O((|S_1| + |S_2|) \log |S_1|)$. As the running time is defined to be $(|S_1| + |S_2|) \cdot \gamma(|S_1| + |S_2|)$, we have $\gamma(|S_1| + |S_2|) \leq \log(|S_1| + |S_2|)$.

Remark. We note the effect of the running time of the separator algorithm (Lemma 12) on the running time of the MIX algorithm: If the running time was $O(n^c)$ for some constant c , then the additive term in the recursion would have increase to $O((|S_1| + |S_2|)^c)$, leading to $\gamma(n) = n^{c-1} \cdot \log n$; such a running time would be sublinear only for $c < 2$; here, by achieving $c = 1$, we get the fastest possible running time which implies the polylogarithmic running time for MIX for fat objects. ◀

6 DISQS for fat objects

In this section, we define DISQS for various classes of geometric objects. First observe that for intervals, a 1-DISQS with running time $O(\log n)$ follows from the classic greedy algorithm [25]. By storing the intervals in an augmented binary search tree, one can insert and delete intervals as well as answer queries of the form “What is the interval entirely to the right of x with the leftmost right endpoint?” As intervals are fat, this implies a $(1 - \epsilon)$ approximate MIS algorithm with running time $O(\frac{1}{\epsilon} \log n)$. This is not new, in the past year a complicated structure via local exchanges appeared in [12] and soon after a much simpler method [21] using a local rebuilding was obtained; we obtain this as part of our more general scheme for fat objects.

This involves two ideas. First, we use a simple greedy offline algorithm that computes an constant-approximate MIS for fat objects. Then we combine this algorithm with a range searching data structure to implement the greedy choice.

A greedy offline approximation algorithm

Given a collection of fat objects, we consider the independent set obtained by sorting them by size, from the smallest to the largest, and adding them greedily to the independent set, provided they do not intersect anything added previously. We refer to this algorithm as the *greedy* algorithm. It was considered in particular by Chan and Har-Peled [17], but was known in special cases before [22]. From the definition of f -fat objects, every successive object returned by the algorithm can intersect at most f disjoint objects that are larger. Hence this simple algorithm yields a constant-factor approximation algorithm for fat objects.

► **Lemma 14.** *For f -fat objects in dimension d , the greedy algorithm returns an $1/f$ -approximate MIS.*

We need to implement this greedy method as the query of a DISQS; that is, it should support insertion and deletion of objects, and the running time of the greedy algorithm must be output-sensitive. This can be done with a slight variation of classic range intersection query structures, where we can insert and delete objects, mark or unmark objects that intersect a given query, and return the largest unmarked object. Thus each item returned by the greedy algorithm is reported after a constant number of range intersection query operations.

The DISQS data structures thus have running times that match those of the underlying range intersection query structures when the query ranges are from the same family of objects as the objects stored: $O(\log^{2d} n)$ for hyperrectangles, $\tilde{O}(n^{1-\frac{1}{d+2}})$ for disks and $\tilde{O}(n^{1-\frac{1}{d(d+1)}})$ for simplices, see the full version.

Summary

The range intersection queries (details in the full version) combined with the greedy algorithm (Lemma 14) gives a DISQS whose running time depends on the range intersection queries, and whose approximation ratio is the inverse of the fatness constant. From Lemma 13 fat objects have a MIX algorithm. Given the MIX, DISQS, and constant $\epsilon > 0$, Theorem 7 yields a dynamic MIS structure, with worst-case update time depending on the range intersection queries, approximation ratio within ϵ of that from the fatness, and with only a constant-size update set per operation. Putting all these pieces together yields Theorem 1.

References

- 1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 114–125, 2019. doi:10.1145/3313276.3316376.
- 2 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Foundations of Computer Science (FOCS)*, pages 400–409. IEEE, 2013. doi:10.1109/FOCS.2013.50.
- 3 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. In *36th International Symposium on Computational Geometry, SoCG*, pages 2:1–2:15, 2020. doi:10.4230/LIPIcs.SoCG.2020.2.
- 4 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, 1997.

- 5 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 815–826, 2018. doi:10.1145/3188745.3188922.
- 6 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1919–1936, 2019. doi:10.1137/1.9781611975482.116.
- 7 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 382–405, 2019. doi:10.1109/FOCS.2019.00032.
- 8 Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 427–436, 2001. URL: <http://dl.acm.org/citation.cfm?id=365411.365496>.
- 9 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 406–423, 2019. doi:10.1109/FOCS.2019.00033.
- 10 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2537–2549, 2021. doi:10.1137/1.9781611976465.150.
- 11 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2 + \epsilon)$ -approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1872–1885, 2019. doi:10.1137/1.9781611975482.113.
- 12 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. *CoRR*, abs/2007.08643, 2020. arXiv:2007.08643.
- 13 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 892–901. SIAM, 2009. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496867>.
- 14 Parinya Chalermsook and Bartosz Walczak. Coloring and maximum weight independent set of rectangles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 860–868, 2021. doi:10.1137/1.9781611976465.54.
- 15 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, pages 178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.
- 16 Timothy M. Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, pages 373–392, 2012.
- 17 Timothy M. Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012.
- 18 Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. In *36th International Symposium on Computational Geometry, SoCG*, page In press, 2021.
- 19 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 370–381, 2019. doi:10.1109/FOCS.2019.00031.
- 20 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Foundations of Computer Science (FOCS)*, pages 820–829. IEEE, 2016. doi:10.1109/FOCS.2016.92.
- 21 Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. *CoRR*, abs/2012.15002, 2020. arXiv:2012.15002.

- 22 Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Comput. Geom.*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0.
- 23 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, pages 1302–1323, 2005. doi:10.1137/S0097539702402676.
- 24 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, pages 133–137, 1981. doi:10.1016/0020-0190(81)90111-3.
- 25 Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1(2):180–187, 1972. doi:10.1137/0201013.
- 26 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, pages 227–242, 2015. doi:10.1016/j.tcs.2014.09.046.
- 27 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 537–550, 2017. doi:10.1145/3055399.3055493.
- 28 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. doi:10.1007/BF02392825.
- 29 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *36th International Symposium on Computational Geometry, SoCG*, pages 51:1–51:14, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.
- 30 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, pages 130–136, 1985. doi:10.1145/2455.214106.
- 31 Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 384–393, 1998. URL: <http://dl.acm.org/citation.cfm?id=314613.314768>.
- 32 Madhav V. Marathe, H. Breu, Harry B. Hunt III, S. S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995. doi:10.1002/net.3230250205.
- 33 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. *CoRR*, abs/2101.00326, 2021. arXiv:2101.00326.
- 34 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.
- 35 Bram Verweij and Karen Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*, pages 426–437, 1999. doi:10.1007/3-540-48481-7_37.
- 36 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, pages 103–128, 2007. doi:10.4086/toc.2007.v003a006.

Balanced Crown Decomposition for Connectivity Constraints

Katrin Casel  

Hasso Plattner Institute, University of Potsdam, Germany

Tobias Friedrich  



Hasso Plattner Institute, University of Potsdam, Germany

Davis Issac  

Hasso Plattner Institute, University of Potsdam, Germany

Aikaterini Niklanovits  

Hasso Plattner Institute, University of Potsdam, Germany

Ziena Zeif  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

We introduce the *balanced crown decomposition* that captures the structure imposed on graphs by their connected induced subgraphs of a given size. Such subgraphs are a popular modeling tool in various application areas, where the non-local nature of the connectivity condition usually results in very challenging algorithmic tasks. The balanced crown decomposition is a combination of a crown decomposition and a balanced partition which makes it applicable to graph editing as well as graph packing and partitioning problems. We illustrate this by deriving improved approximation algorithms and kernelization for a variety of such problems.

In particular, through this structure, we obtain the first constant-factor approximation for the BALANCED CONNECTED PARTITION (BCP) problem, where the task is to partition a vertex-weighted graph into k connected components of approximately equal weight. We derive a 3-approximation for the two most commonly used objectives of maximizing the weight of the lightest component or minimizing the weight of the heaviest component.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases crown decomposition, connected partition, balanced partition, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.26

Related Version *Full Version:* <https://arxiv.org/abs/2011.04528>

1 Introduction

Connected subgraphs are one of the most natural structures to encode aspects of a practical task, modeled as a graph problem. On the one side, such subgraphs represent structures we seek to discover, such as territories for postal delivery and similar districting problems (see e.g. the survey [22]). From another perspective, the structures of interest could be operations that scatter a graph into small connected components; a structure e.g. used to model vulnerability in network security (see e.g. the survey [2]). Partitioning a graph into connected components of a given size is also used as a model for task allocation to robots [43]. From an algorithmic perspective, connectivity is a non-local requirement which makes it particularly challenging. We introduce a graph structure that can be used to design efficient algorithms for a broad class of problems involving connected subgraphs of a given size.



© Katrin Casel, Tobias Friedrich, Davis Issac, Aikaterini Niklanovits, and Ziena Zeif;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 26; pp. 26:1–26:15

Leibniz International Proceedings in Informatics



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

One useful structure to derive information about connected subgraphs is a *connected partition*, defined as follows. Given a graph $G = (V, E)$, a connected partition of G is a partition V_1, \dots, V_k of V such that the graph induced by the vertex set V_i is connected for each $1 \leq i \leq k$, where $k \in \mathbb{N}$ is the size of the partition. Often, we are not interested in just any connected partition but in those that have the additional property of being *balanced*. Informally speaking, a connected partition is considered balanced if the sets V_i have approximately equal cardinality. There are several measures to assess the quality of a *balanced connected partition* (BCP for short), with the two most commonly used objectives being to maximize $\min_{1 \leq i \leq k} |V_i|$ or to minimize $\max_{1 \leq i \leq k} |V_i|$, as first introduced for trees in [30] and [25], respectively. Despite extensive studies on these problems for the past 40 years, see e.g. [7, 30, 32, 35, 37, 38, 39] the best known approximation ratio for the Min-Max objective depends on the number of sets k [11]. For the Max-Min objective not even such a result is known; only for the cases with k restricted to 2 or 3, there exist approximations with ratio $\frac{4}{3}$ [12] and $\frac{5}{3}$ [8], respectively. Deriving an approximation with a ratio independent of k seems to require a new strategy.

Helpful structures for both of the objectives Max-Min and Min-Max, as a sort of compromise, are BCP's such that $\lambda_l \leq |V_i| \leq \lambda_r$ for some fixed bounds λ_l, λ_r . We call this compromise structure $[\lambda_l, \lambda_r]$ -CVP (connected *vertex* partition), and it is one ingredient of *balanced crown decomposition*, the main structural object that we present. In the case that no $[\lambda_l, \lambda_r]$ -CVP exists for a graph, we can learn something about its structure. In particular, our *balanced crown decomposition theorem* (Theorem 7) shows that for any $k, \lambda > 0$, the non-existence of a $[\lambda, 3\lambda - 3]$ -CVP implies the existence of a vertex set $H \subseteq V$ of cardinality at most k that disconnects at least one component of size less than λ from G . Such sets H are in a sense the dual of balanced connected partitions.

Small subsets of vertices that disconnect a graph are usually called *vertex separators*, and they are one of the most powerful tools for designing efficient graph algorithms. In a sense, they are the base requirement of successful divide-and-conquer strategies for graph problems. This generality and their wide applicability has made the study of separators a rich and active research field, see e.g. the book by Rosenberg and Heath [31], or the line of research initialized by the seminal paper of Lipton and Tarjan [27] on separators in planar graphs. Numerous different types of separator structures emerged over the past couple of decades. In the context of connectivity problems, the separator structures of particular interest are crown decompositions; a classical tool to derive kernelizations in the field of parameterized complexity. We refer to chapter 4 of the book on kernelization [18] for more details on crown decompositions and their applications.

Crown decomposition was introduced as a generalization of Hall's Theorem in [13]. More precisely, a *crown decomposition* of a graph $G = (V, E)$ is a partition of V into three sets H (*head*), C (*crown*) and R (*royal body*), such that H separates C from R , C is an independent set in G , and there exists a matching of size $|H|$ among the edges $E \cap (H \times C)$. Notice that the set H is the separator set, and the property of C being an independent set can be seen as H splitting connected components of size 1 from the graph. The condition of the matching from H into C models a trade-off between the size of the separator and the amount of small sets that are separated. Different versions of crown decompositions have been introduced in the literature, adjusting the structure to specific application scenarios.

The structure of particular interest to us is the *q-Weighted Crown Decomposition* introduced by Xiao [40]. Here, the crown C is no longer an independent set, but has the restriction that each connected component in it has size at most q (generalizing the notion of independent set for $q = 1$); and there exist an assignment of connected components of

the crown to the head such that each vertex in the head is assigned at least q vertices. This assignment generalizes the notion of matching in the original crown decomposition. Such a weighted crown decomposition can be derived using the *Expansion Lemma* as stated in [17, Chapter 5.3], and its generalization to the *Weighted Expansion Lemma* as given in slightly different forms in [24] and [40]. The *expansions* derived by these lemmas can be thought of as bipartite analogues of the crown decomposition. Formally, given a bipartite graph $G = (A, B, E)$, a q -(Weighted) Expansion is given by sets $H \subseteq A$ and $C \subseteq B$ such that the neighborhood of C is contained in H and an assignment $f: C \rightarrow H$ such that the number (resp. weight, in the vertex-weighted case) of vertices assigned to each vertex in H is at least q (resp. $q - W + 1$ where W is the largest weight). Both Kumar and Lokshtanov [24] and Xiao [40] use their respective Weighted Expansion Lemma to derive kernels for COMPONENT ORDER CONNECTIVITY, a version of the editing problem that we also consider in a more general form under the name W -WEIGHT SEPARATOR.

To create our new structure *balanced crown decomposition*, we combine balanced connected partitions and crown decompositions to derive a tool that has the advantages of both of the individual structures. Essentially, it is a weighted crown decomposition with the additional property that the body has a balanced connected partition. Also, note that we allow a more generalized version of weighted crown decomposition than Xiao [40], by considering weighted vertices. Formally, we consider *vertex-weighted* graphs $G = (V, E, w)$ with integer-weights $w: V \rightarrow \mathbb{N}$. For simplicity we use $w(V') = \sum_{v \in V'} w(v)$ for the *weight of a subset* $V' \subseteq V$.

We show that balanced crown decompositions have applications for various kinds of problems involving connectivity constraints. Specifically we discuss for the three types editing, packing and partitioning the following problems on input $G = (V, E, w)$ and $k, W \in \mathbb{N}$:

MAX-MIN (MIN-MAX) BCP: Decide if there exists a connected partition V_1, \dots, V_k of V such that $w(V_i) \geq W$ (resp. $w(V_i) \leq W$) for each $i \in [k]$; usually stated as optimization problem to maximize/minimize W .

W -WEIGHT SEPARATOR: Decide if there exists a set $S \subseteq V$ with $|S| \leq k$ whose removal from G yields a graph with each connected component having weight less than W .

W -WEIGHT PACKING: Decide if there exist k pairwise disjoint sets $V_1, \dots, V_k \subseteq V$ with $w(V_i) \geq W$, such that the graph induced by V_i is connected, for each $i \in [k]$.

We remark that the problems W -WEIGHT SEPARATOR and W -WEIGHT PACKING have been studied mostly on the unweighted versions, also known as COMPONENT ORDER CONNECTIVITY and T_r -PACKING, respectively.

For all results of this paper, we consider the RAM model of computation with word size $O(\log(|V| + \max_{v \in V} w(v)))$. All our algorithms are polynomial w.r.t. the encoding of input.

Lastly, we point out that this short version of the paper contains only a summary of how to compute a balanced crown decomposition and its applications. Here we provide descriptions of the algorithms used and proof sketches about the results achieved through this structure. For technical details and complete proofs, we refer to the full version of the paper.

1.1 Our Contribution

Our main contributions can be summarized as follows:

1. **Balanced Crown Decomposition (BCD):** The main contribution of our paper is a new crown decomposition tailored for problems with connectivity constraints. Our novel addition over previous crown decompositions is that we also give a partition of the *body* into connected parts of roughly similar size. More precisely, we divide the graph into

C , H , and R such that C, H, R is a weighted-crown decomposition and also a $[\lambda, 3\lambda]$ -CVP is given for R . Definition 6 gives the formal definition of BCD, and Theorem 7 gives our main result about computing BCD. We believe that apart from the applications used in this paper, BCD will find applications for other problems with connectivity constraints.

2. **Balanced Expansion:** We also give a novel variation of the expansion lemma, which is an important constituent of our algorithm for BCD. Given a bipartite graph $G = (A, B, E)$, we give an expansion with $H \subseteq A, C \subseteq B$, with the addition that the expansion f while being a weighted q -expansion from C to H , now also maps $B \setminus C$ to neighbors in $A \setminus H$ such that only a bounded weight is assigned to each vertex in $A \setminus H$. See Definition 1 for a more formal definition and Theorem 2 for our result on computing balanced expansion. Apart from its usage here to compute BCD, the balanced expansion could be of independent interest, given the significance of the Expansion Lemma in parameterized complexity.
3. **Approximation algorithms for BCP:** Using BCD, we give 3-approximation algorithms for both MAX-MIN and MIN-MAX BCP. These are the first constant approximations for both problems in polynomial time for a general k . Recall that despite numerous efforts in the past 40 years, only a $k/2$ -approximation for MIN-MAX BCP [11], and constant-factor approximations for the particular cases of $k = 2, 3$ for MAX-MIN BCP [12, 8] were known.
4. **Improved Kernels for W-weight separator and packing:** BCD directly gives a $3kW$ -kernel for both of the problems improving over the previous best polynomial time kernels of size $9kW$ [40] and $\mathcal{O}(kW^3)$ [9]. Especially, we get the same improvements for the unweighted versions COMPONENT ORDER CONNECTIVITY and T_r -PACKING.
5. **Faster algorithms for Expansion:** Our algorithm for Balanced Expansion, also gives an alternative flow-based method for computing the standard (weighted) expansion. Our algorithm can compute a (weighted) expansion in $\mathcal{O}(|V||E|)$ surpassing the previous best runtimes of $\mathcal{O}(|V|^{1.5}|E|)$ and $\mathcal{O}(|E||V|^{1.5}W^{2.5})$ [18, Chapter 5.3] (here W is the largest weight) for unweighted and weighted expansion, respectively. In particular, for weighted expansion, our runtime does not depend on the weights and is the first algorithm that runs in time polynomial w.r.t. the length of the input-encoding. The improvement in runtime may turn out to be useful to speed up kernelizations for other problems.

1.2 Related work

Both variants of BCP were first introduced for trees, where MAX-MIN BCP and MIN-MAX BCP are introduced in [30] and [25], respectively. For this restriction to trees, a linear time algorithm was provided for both variants in [19]. For both variants of BCP, a Δ_T -approximation is given in [3] where Δ_T is the maximal degree of an arbitrary spanning tree of the input graph; for MAX-MIN BCP the result holds only when the input is restricted to weights with $\max_{v \in V} w(v) \leq \frac{w(G)}{\Delta_k}$. Also, although not explicitly stated, a $(1 + \ln(k))$ -approximation in $\mathcal{O}(n^k)$ time for MIN-MAX BCP $_k$ follows from the results in [10]. With respect to lower bounds, it is known that there exists no approximation for MAX-MIN BCP with a ratio below $6/5$, unless $P \neq NP$ [7]. For the unweighted case, i.e. $w \equiv 1$, the best known result for MIN-MAX BCP is the $\frac{k}{2}$ -approximation for every $k \geq 3$ given in [11].

Balanced connected partitions are also studied for particular cases of k , denoted BCP $_k$. The restriction BCP $_2$, i.e. balanced connected bipartition, is already **NP**-hard [5]. On the positive side, a $\frac{4}{3}$ -approximation for MAX-MIN BCP $_2$ is given in [12], and in [11] this result is used to derive a $\frac{5}{4}$ -approximation for MIN-MAX BCP $_2$. For tripartition, approximations for MAX-MIN BCP $_3$ and MIN-MAX BCP $_3$ with ratios $\frac{5}{3}$ and $\frac{3}{2}$, respectively, are given in [8].

BCP in unit-weighted k -connected graphs can be seen as a special case of the Győri-Lovász Theorem (independently given by Győri [20] and Lovász [28]). It states that for any k -connected graph $G = (V, E)$ and integers n_1, \dots, n_k with $n_1 + \dots + n_k = |V|$, there exists

a connected partition V_1, \dots, V_k of V with $|V_i| = n_i$ for all $i \in [k]$. Moreover, it is possible to fix vertices v_1, \dots, v_k and request $v_i \in V_i$ for all $i \in [k]$. The Győri-Lovász Theorem is extended to weighted directed graphs in [10] and Győri's original proof is generalized to weighted undirected graphs in [6]. Polynomial algorithms to also compute such connected partitions are only known for the particular cases $k = 2, 3, 4$ [32, 35, 21] and all $k \geq 5$ are still open. A restricted case of BCP where the partitions are allowed to differ only by a size of one, has been studied from the FPT viewpoint [15].

W -WEIGHT SEPARATOR occurs in the literature under different names. The unweighted version is studied under the names p -Size Separator [40] or ℓ -Component Order Connectivity (COC) [14, 24]; where $p, \ell = W - 1$ translate this to our definition of W -WEIGHT SEPARATOR with unit weights. In [14] a weighted version of COC denoted by *Weighted Component Order Connectivity* (w COC) is introduced. This problem differs from our W -WEIGHT SEPARATOR by searching for a set S with $w(S) \leq k$ instead of $|S| \leq k$.

Note that W -WEIGHT SEPARATOR with $W = 2$ and unit weights yields the classical problem VERTEX COVER. This in particular shows that W (alone) is not a suitable parameter from the FPT viewpoint. Further, W -WEIGHT SEPARATOR is $\mathbf{W}[1]$ -hard for parameter k , even when restricted to split graphs [14]. These lower bounds lead to studying parameterization by $W + k$. Stated with $\ell = W - 1$, a kernel of size $9k\ell$ is given in [40]. Also [24] derives a kernel of size $2k\ell$ in time $\mathcal{O}(|V|^\ell)$. Both of these results are for unit weights. An $\mathcal{O}(k\ell(k + \ell)^2)$ weight kernel for the related problem w -COC is given in [14].

For $W = 3$ and unit weights, W -WEIGHT SEPARATOR corresponds to VERTEX COVER P_3 or 3-PATH VERTEX COVER (see e.g. [34] and [4]), first studied by Yannakakis [42] under the name DISSOCIATION NUMBER. The best known kernel for this problem is of size $5k$ and given in [41]. W -WEIGHT PACKING with unit weights is equivalent to T_r -PACKING with $r = W + 1$, where T_r is a tree with at least r edges, as defined in [9]; note that any connected component with at least W vertices has at least $r - 1$ edges, and any tree with $r - 1$ edges has exactly W vertices. The best known kernel for this problem is of size $\mathcal{O}(kW^3)$ by [9].

W -WEIGHT PACKING is also studied for particular values of W . The case $W = 2$ with unit weights is equivalent to the MAXIMUM MATCHING problem; note that a matching of size k can be derived from a solution V_1, \dots, V_k for 2-WEIGHT PACKING by choosing arbitrarily any edge in a set V_i with $|V_i| > 2$. In a similar way, the particular case of $W = 3$ is a problem studied under the names P_2 -PACKING or PACKING 3-VERTEX PATHS (see e.g [36] and [23]). A $5k$ kernel for this problem is given in [26].

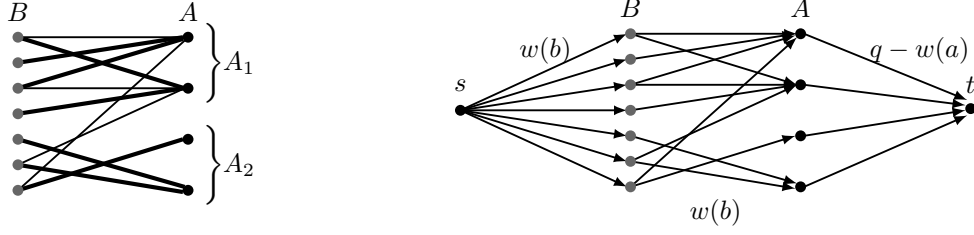
2 Balanced Expansion

In this section we introduce a balanced generalization of weighted expansions that we call *balanced expansion*.

Balanced expansion extends the existing weighted expansion structures and is one of the ingredients to derive our main BCD structure in the next section. Like the weighted expansion, it is a structure on bipartite graphs. We write $G = (A \cup B, E, w)$ for bipartite vertex-weighted graphs, where $w: A \cup B \rightarrow \mathbb{N}$ is its weight function. See Figure 1 for an illustration of this structure.

► **Definition 1** (*balanced expansion*). Let $G = (A \cup B, E, w)$ be a bipartite vertex-weighted graph, where $w_{\max}^B = \max_{b \in B} w(b)$. For $q \in \mathbb{N}_0$, a partition $A_1 \cup A_2$ of A and $f: B \rightarrow A$, the tuple (A_1, A_2, f, q) is called a *balanced expansion* if:

1. $w(a) + w(f^{-1}(a)) \begin{cases} \geq q - w_{\max}^B + 1, & a \in A_1 \\ \leq q + w_{\max}^B - 1, & a \in A_2 \end{cases}$
2. $f(b) \in N(b)$
3. $N(f^{-1}(A_1)) \subseteq A_1$



■ **Figure 1** Left: Balanced expansion for $w(b) = 1$ for all $b \in B$, $q = 2$ and assignment f depicted with bold edges. Right: Flow network embedding of the graph on the left.

Our main result of this section is the following theorem.

► **Theorem 2 (balanced expansion).** Consider a vertex-weighted bipartite graph $G = (A \cup B, E, w)$ with no isolates in B , and $q \geq \max_{b \in B} w(b) = w_{\max}^B$. A balanced expansion (A_1, A_2, f, q) for G can be computed in $\mathcal{O}(|V||E|)$ time. Furthermore, if $w(A) + w(B) \geq q|A|$, then $A_1 \neq \emptyset$.

As intermediate structure we use a fractional version of the balanced expansion where we partially assign weights from vertices of B to vertices of A encoded as edge weights.

► **Definition 3 (fractional balanced expansion).** Let $G = (A \cup B, E, w)$ be a bipartite vertex-weighted graph. For $q \in \mathbb{N}_0$, a partition $A_1 \cup A_2$ of A and $g: E \rightarrow \mathbb{N}_0$, the tuple (A_1, A_2, g, q) is called fractional balanced expansion if:

1. $w(a) + \sum_{b \in B} g(ab) \begin{cases} \geq q, & a \in A_1 \\ \leq q, & a \in A_2 \end{cases}$
2. $\forall b \in B: \sum_{a \in A} g(ab) \leq w(b)$ (capacity)
3. $N(B_U \cup B_{A_1}) \subseteq A_1$ (separator)
where $B_a := \{b \in B \mid g(ab) > 0\}$ for $a \in A$, $B_{A'} := \bigcup_{a \in A'} B_a$ for $A' \subseteq A$ and $B_U := \{b \in B \mid \sum_{a \in A} g(ab) < w(b)\}$

We prove a fractional version of our result first in the following lemma.

► **Lemma 4 (fractional balanced expansion).** Given a vertex-weighted bipartite graph $G = (A \cup B, E, w)$ with no isolated vertices in B and $q \in \mathbb{N}_0$, a fractional balanced expansion (A_1, A_2, g, q) can be computed in $\mathcal{O}(|V||E|)$. Also, if $w(A) + w(B) \geq q|A|$, then $A_1 \neq \emptyset$.

Proof Sketch. The main idea is to embed G to a capacitated flow network in a standard way (see Figure 1). We construct a network $H = (A \cup B \cup \{s, t\}, \vec{E}, c)$. To obtain H from G , add source s and sink t , and arcs \vec{E} with a capacity function $c: \vec{E} \rightarrow \mathbb{N}$ defined as follows. For every $b \in B$, add an arc $\vec{s}b$ with capacity $w(b)$ and for every $a \in A$, add an arc $\vec{a}t$ with capacity $q - w(a)$. Moreover, transform every edge $ab \in E$ to an arc $\vec{b}a$ with capacity $w(b)$.

We compute a max flow $f: \vec{E} \rightarrow \mathbb{N}$ and define the saturated vertices $A' \subseteq A$ as $a \in A$ with $f(\vec{a}t) = c(\vec{a}t)$. We now gradually build the sets A_1 and A_2 . The vertices of A' are potential vertices for A_1 while the unsaturated vertices are immediately added to A_2 . We define $F := \sum_{\vec{e} \in \delta^-(t)} f(\vec{e})$ as the flow value, where $\delta^-(v)$ denotes the incoming, and $\delta^+(v)$ the outgoing arcs for $v \in V(G)$. The final selection of A_1 follows by individually increasing the capacity by one for each $\vec{a}t$ for $a \in A'$, and checking whether the flow value increases by computing a new max flow f_a with the increased capacity of $\vec{a}t$. Let F_a be the flow value when the capacity of $\vec{a}t$ is increased by one. If $F_a > F$, then the vertex is added to A_1 , otherwise it is added to A_2 . The intuition behind this selection can be explained as

follows: first observe that each $b \in B$ that has an edge ba_2 to some $a_2 \in A_2$ is saturated, i.e. $\sum_{\vec{e} \in \delta^+(b)} f(\vec{e}) = w(b)$. Otherwise, we could route an additional unit of flow from b to a_2 either in f or in f_{a_2} , giving a contradiction to the fact that $a_2 \in A_2$. Consequently, every unsaturated $b \in B$ is adjacent only to A_1 . The second observation is that there are no $b \in B$ with $f(\vec{ba_1}) > 0$ and $ba_2 \in \vec{E}$ for $a_1 \in A_1$ and $a_2 \in A_2$. If such a b exist, we show that we can route an extra unit of flow from b to a_2 either in f or f_{a_2} . The idea is that we could reroute one unit of flow from $\vec{ba_1}$ to $\vec{ba_2}$ creating a vacuum for one unit of incoming flow in a_1 . Since f_{a_1} routed one unit flow more than f , we could use a similar flow routing as in f_{a_1} to fill this vacuum, thus contradicting the maximality of either f or f_{a_2} . As a result, all vertices added to A_1 have the desired exclusive neighborhood in B encoded by f . Finally, in order to derive g we convert the flow arc values of f to edge weights for g . Note that the required upper bound on the assignment of A_2 follows from the capacities of the arcs from A to t , and the required lower bound on the assignment of A_1 follows from the vertices in A_1 being saturated. Regarding running time, we remark that it is sufficient to find one max-flow f at the beginning and then computing each f_a with only one augmenting flow step. The max-flow f can be computed in $\mathcal{O}(|V||E|)$ time using the algorithm by Orlin [29]. ◀

Proof Sketch of Theorem 2. Once we have the fractional balanced expansion g , our first step is modifying the edge weights g such that the edge-weighted graph $G' := (V, \{ab \in E | g(a, b) > 0\}, g)$ becomes a forest, without changing the sum $\sum_{b \in N(a)} g(a, b)$ for any $a \in A$ and at the same time ensuring that $\sum_{a \in N(b)} g(a, b) \leq w(b)$ for all $b \in B$. This is possible through a standard cycle canceling process. Now consider the trees in this forest. The trees intersecting A_1 are disjoint from the trees intersecting A_2 due to the separation property of the balanced fractional expansion. For a tree T intersecting A_1 , we allocate each $b \in V(T) \cap B$ completely to its parent in T . This way, any $a \in V(T) \cap A_1$ loses at most the assignment from its parent and hence its assignment decreases by at most $w_{\max}^B - 1$ as required. Now consider a tree T intersecting A_2 . If a $b \in V(T) \cap B$ is a leaf of T its assignment has to be non-fractional, so it can be completely assigned to its parent a and deleted from the tree. This way, all leafs can be assumed to be from A_2 . We then allocate each $b \in V(T) \cap B$ to one of its children, and thus to every $a \in V(T) \cap A_2$ at most the assignment from its parent is added, and hence the assignment increases by at most $w_{\max}^B - 1$ as required. ◀

Before moving to BCD, we formally state the aforementioned implication of the results in this section on the runtime of computing (non-balanced) expansions.

► **Lemma 5** (Weighted Expansion Lemma). *Let $G = (A \cup B, E)$ be a bipartite graph without isolated vertices in B , $w: B \rightarrow \{1, \dots, W\}$, and $q \in \mathbb{N}_0$. A q -weighted expansion (f, H, C) in G can be computed in time $\mathcal{O}(|A \cup B||E|)$. Furthermore, if $w(B) \geq q|A|$ then $H \neq \emptyset$.*

3 Balanced Crown Decomposition

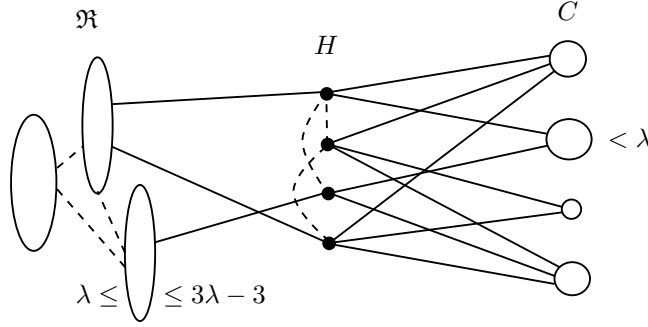
In this section we introduce our combination of balanced connected partition and crown decomposition that we call *balanced crown decomposition*, formally defined as follows (see also Figure 2 for an illustration).

► **Definition 6.** A λ -balanced crown decomposition (λ -BCD) of a vertex-weighted graph $G = (V, E, w)$ is a tuple (C, H, \mathfrak{R}, f) , where $\{H, C, R\}$ is a partition of V , the set \mathfrak{R} is a partition of R , and $f: \mathbb{CC}(C) \rightarrow H$ where $\mathbb{CC}(C)$ is the set of connected components of $G[C]$, such that:

1. there are no edges from C to R
 2. $w(Q) < \lambda$ for each $Q \in \mathcal{CC}(C)$
 3. $f(Q) \in N(Q)$ for each $Q \in \mathcal{CC}(C)$
 4. $w(h) + w(f^{-1}(h)) \geq \lambda$ for each $h \in H$
 5. $G[R']$ is connected and $\lambda \leq w(R') \leq 3\lambda - 3$ for each $R' \in \mathfrak{R}$.
- (weighted crown dec.)

Our novel contribution is condition 5, that gives a balanced connected partition of the *body*. Without this condition, the structure is same as the weighted crown decomposition [40]. Observe that if there is a connected component of weight less than λ in G , then there is no λ -balanced crown decomposition for G . In the applications of BCD, such small components in the input are usually removed through some form of preprocessing.

We point out that the ratio 3 between the upper and lower bound in condition 5 of BCD is not arbitrary, but the best possible, since we want to ensure the existence of this structure in case all connected components have weight at least λ . A simple tight example is a triangle with each vertex having a weight of $\lambda - 1$; here, $C = H = \emptyset$ is the only possibility and hence $\mathfrak{R} = \{V\}$ is the only possible partition of $R = V$.



■ **Figure 2** λ -balanced crown decomposition.

The main structural result of the paper is the following.

► **Theorem 7** (Balanced Crown Decomposition Theorem). *Let $G = (V, E, w)$ be a vertex-weighted graph and $\lambda \in \mathbb{N}$, such that each connected component in G has weight at least λ . A λ -balanced crown decomposition (C, H, \mathfrak{R}, f) of G can be computed in $\mathcal{O}(k^2 |V| |E|)$ time, where $k = |H| + |\mathfrak{R}| \leq \min\{w(G)/\lambda, |V|\}$.*

The proof of this result is very technical and we therefore here only give a very high-level overview of the ideas. Observe that the condition $|H| + |\mathfrak{R}| \leq \min\{w(G)/\lambda, |V|\}$ holds, since $\{\{h\} \cup f^{-1}(h) : h \in H\} \cup \mathfrak{R}$ is a partition of the vertices with each part having weight at least λ . This bound is also used to track our progress in our BCD algorithm. We maintain a set H that can be thought of as a *potential head* (not necessarily a separator), a set of connected components of weight smaller than λ (some of them assigned to vertices in H by a partial assignment f) which can be thought of as a *potential crown*, and a remaining *body* that is packed according to condition 5.

To easily talk about condition 5 in the following, we say \mathcal{U} is a *connected packing* in $V' \subseteq V$, if for every $U \in \mathcal{U}$ we have $U \subseteq V'$, U induces a connected subgraph in G and $\bigcup_{U \in \mathcal{U}} U = \emptyset$. We say \mathcal{U} is an $[a, b]$ -*connected packing* of V' if $w(U) \in [a, b]$ for every $U \in \mathcal{U}$ and that \mathcal{U} is *maximal* if the remaining graph does not have a connected component of weight at least a . Recall that we say \mathcal{U} is a CVP or $[a, b]$ -CVP of V' if additionally $\bigcup_{U \in \mathcal{U}} U = V'$ holds, and observe that condition 5 asks for a $[\lambda, 3\lambda - 3]$ -CVP of the body R .

Proof Sketch of Theorem 7. Let $G = (V, E, w)$ be a vertex-weighted graph and $\lambda \in \mathbb{N}$ such that each connected component of G has weight at least λ . We reduce G by deleting all vertices of weight more than λ and all connected components of size smaller than λ that occur after this deletion. Suppose we have a λ -BCD for the reduced graph, then a λ -BCD of G can be built by adding the deleted heavy vertices to the head, the deleted small components to the crown and assigning (by the function f in the definition of λ -BCD) each of these components arbitrarily to a heavy vertex it is adjacent to. Thus, we can assume that every vertex has weight at most λ . See Figure 3 for an illustration of the structures arising below.

We start with a maximal $[\lambda, 2\lambda]$ -connected packing of G which is obtained greedily (slight deviation from the main proof to provide better intuition). Let $\mathfrak{R} = \{R_1, R_2, \dots\}$ be this packing, C be the vertices not in the packing, and let $\mathbb{CC}(C) = \{C_1, C_2, \dots\}$ be the connected components of $G[C]$. Note that by the maximality of the packing, $w(C_i) < \lambda$ for all i . Think of \mathfrak{R} as the current body and C as the current crown, and the head is empty in the beginning. Note that at this point we do not ensure that there are no edges between crown and body. If C is empty then we already have a λ -BCD (with empty crown and head). Also, if we can somehow assign each C_i to some adjacent R_i such that each R_i is assigned weight at most 3λ (including its own weight), then we have also built a λ -BCD (with empty crown and head). Assuming neither of these cases hold, there has to exist an R_i such that its weight plus the weight of the neighborhood in the crown part is at least 3λ ; recall that we assumed that all connected components of G have weight at least λ , so each C_j is connected to at least one component in \mathfrak{R} . We call the subgraph induced by R_i together with all C_j that are connected to it the *effective neighborhood of R_i* , and its weight the *effective weight of R_i* .

In case we have not found a λ -BCD yet, we pick an R_i with effective weight at least 3λ and use the following fact derived from the famous results of Tarjan [33, 16]: for any connected graph of total weight at least 3λ and largest vertex weight at most λ , we can efficiently either find a partition of it into two connected subgraphs of weight at least λ each, or find a cut-vertex that cuts the graph into components each of weight less than λ . If the effective neighborhood is divided into two, we take each of the two parts into the body and remove R_i , thus increasing the body size by one. In the other case, that is, if we find a cut-vertex c , then we add c to the head and the components of $R_i \setminus \{c\}$ (each having weight less than λ) to the crown $\mathbb{CC}(C)$. We assign with a *partial* function $f: \mathbb{CC}(C) \rightarrow H$ some of these components that we just added to $\mathbb{CC}(C)$ to c such that c is assigned a total weight of at least 3λ (including weight of c). The reason for assigning 3λ when we only require λ by the definition of BCD, will become clear in the following. The new components added to the crown could have edges to the old components there and hence can merge with these. If at any point it happens that there is a component of weight at least λ in the crown, then we immediately add it to the body. This could cause some head vertex to loose some of its assignment, but since it had 3λ assigned to it, an assignment of at least λ remains. This is because we ensure that the part we move to the body can have weight at most 2λ , as we move it immediately as the weight is at least λ , and each addition is by steps of less than λ .

We repeat this process of picking an effective neighborhood of an R_i and dividing or cutting it. We point out that when we calculate effective neighborhoods and weights, we do not consider the crown parts that are already assigned by f . This process continues until the effective weights of all sets R_i are less than 3λ . We claim that the reason why we have not arrived at a λ -BCD yet could only be that there are crown parts that do not have edges to the body (we call them *private components*) and *not* assigned by f , while there are also crown parts that have edges to the body (non-private components) and assigned by f . Note that if there are no unassigned private components, we can merge all unassigned crown

parts with some set in the body and create a $[\lambda, 3\lambda]$ -CVP given by the R_i 's and the sets $\{v\} \cup f^{-1}(v)$. Note that since the effective weights were lighter than 3λ , the body parts after the merging are lighter than 3λ . Also, if f does not assign any non-private components, we can assign unassigned private components to arbitrary neighbors in H , and merge unassigned non-private components to body obtaining a λ -BCD.

We modify the assignment f to switch non-private with private components in the best way possible. For this we use the balanced expansion Theorem 2. We build the bipartite graph where the set A are the head vertices, and the set B are the private crown components each contracted into a single vertex. Theorem 2 with expansion parameter 2λ then gives us sets $A' \subseteq A$ and $B' \subseteq B$ and an assignment f' such that $w(\{a'\} \cup f'^{-1}(a')) \geq \lambda$ for all $a' \in A'$ and $w(\{a\} \cup f'^{-1}(a)) \leq 3\lambda$ for all $a \in A \setminus A'$, and the crown components in B' are completely assigned to A' and do not have neighbors in $A \setminus A'$. Note that since B was the set of private components, the components B' do not have neighbors in the body either. Now augment f' by assigning to $A \setminus A'$ also enough non-private components such that they have an assignment of at least 3λ each. This is possible since each vertex in $A \setminus A'$ has an assignment of 3λ by f which did not use any components from B' (as there are no edges from B' to $A \setminus A'$). Note that this augmentation of f' needs to be done carefully since the private components could be assigned by the balanced expansion differently than by f .

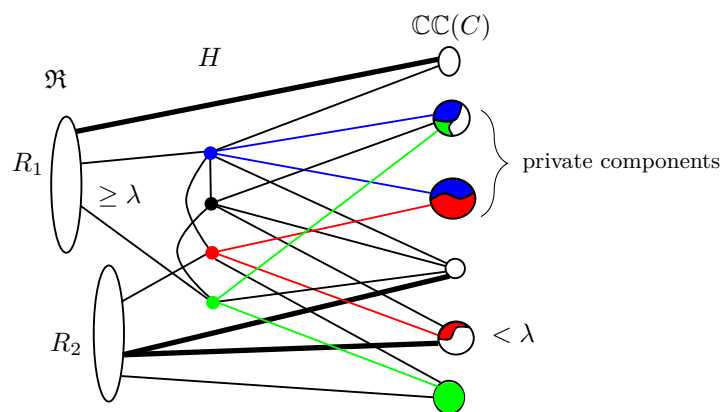
By f' all private components are now assigned to the head, but there could still be non-private ones assigned as well. But now, if the effective weight of each R_i is at most 3λ , we can add the unassigned crown parts to sets in \mathfrak{R} , and thus create a λ -BCD: A' with its assignment by f' are head and crown, and \mathfrak{R} plus the sets $\{a\} \cup f'^{-1}(a)$ with $a \in A \setminus A'$ are a $[\lambda, 3\lambda]$ -CVP of the body. Thus, if we are not successful, there exists an R_i with effective weight more than 3λ and we continue by dividing or cutting it. Note that we can proceed with f' replacing f although some head vertices (from what was A' in the balanced expansion) might only have weight λ assigned to them (and not 3λ), because the crown parts assigned to them are private and hence do not interfere with the further process.

To analyse the run time, we estimate how often we divide or cut a set R_i ; note that each such step can be performed in $\mathcal{O}(|V||E|)$. Throughout our algorithm, the value $|H| + |\mathfrak{R}|$ is non-decreasing, and upper bounded by k . Every time we divide some R_i , we increase $|H| + |\mathfrak{R}|$, hence this happens at most k times. Every time we cut some R_i we increase $|H|$ by one. Since $|H|$ is also upper bounded by k , and we are careful not to decrease $|H|$ with the balancing step in-between cut steps, we arrive at a total of at most k^2 divide or cut steps.

One pitfall here is that after applying the balanced expansion one might be tempted to just take A' and its assignment via f' into head and crown respectively, delete it, and start over on the rest of the graph. The problem with this is that we are not guaranteed to find a non-empty set A' (since the private components might not have weight at least $\lambda|A|$). The way we augment f' , we ensure that we retain the preliminary crown, head and body structure, and with this especially the value $|H|$, and can split up another R_i to either increase $|H|$ or $|H| + |\mathfrak{R}|$. Further, the reason why *we cannot use the standard weighted expansion lemma* here is that we would need a lower bound of at least $\lambda|A|$ on the weight of B for this. We cannot ensure that the private components of the crown alone have a weight of at least $\lambda|A|$, since we also used the non-private components for the assignment f .

One detail that we did not mention so far is that it is not possible to assign exactly 3λ to each head vertex. Since the step size we can guarantee is only λ , we might have to assign $(4\lambda - 1)$ in order to get a value of at least 3λ . Recall that we assign a collection of components of weight less than λ . Without further work, this only yields an upper bound of 4λ instead of 3λ for the packing of the body, worsening the quality of our structure (we for

Another technical detail we skipped is that in the assignment f we maintain, the crown parts we map may not be whole sets in $\mathbb{CC}(C)$ (connected components induced by crown vertices). They are connected, but can be subgraphs of some $C_i \in \mathbb{CC}(C)$. We call such subgraphs *sub-components*. Different sub-components of some C_i can be assigned to different heads. Also, for a C_i some of its sub-components can be assigned while others are not. For our structure to converge to a λ -BCD, sub-components have to be classified as private or non-private based on the set $\mathbb{CC}(C)$ they are a part of, so it can happen that a sub-component is non-private but has no edge to the body. Whenever we make the move from crown to body, we therefore have to do a merging of some sub-components such that for each $C_i \in \mathbb{CC}(C)$ either all its sub-components are assigned to the head or none of them are. \blacktriangleleft



4 Applications of Balanced Crown Decomposition

For the problems W -WEIGHT SEPARATOR and W -WEIGHT PACKING we immediately get the following theorems by reducing an instance (G, k, W) by first finding a W -BCD (C, H, \mathfrak{R}, f) of G , and then applying the standard crown reduction rule that removes the head H and crown C from G . We emphasize that the balanced connected partition of the body is crucial to obtain the kernel sizes. These are the first kernels for vertex-weighted graphs, while also improving the state-of-the-art results for the unweighted cases.

► **Theorem 9.** *W-WEIGHT PACKING admits a kernel of weight $3k(W-1)$. Furthermore, such a kernel can be computed in time $\mathcal{O}(k^2|V||E|)$.*

For the optimization variant of the W -WEIGHT PACKING problem, i.e. maximizing the size of packing, the fact that the partition \mathfrak{R} is a solution also gives a 3-approximation; to the best of our knowledge, the first approximation result for the problem.

► **Theorem 10.** *A 3-approximation for the optimization problem of W -WEIGHT PACKING can be computed in $\mathcal{O}(k^{*2}|V||E|)$, where k^* denotes the optimum value.*

To better sketch the ideas for our results for the BCP problems, we denote by I -CVP $_k$ for an interval I , a CVP with k parts where each part has a weight in I . We derive the following result for MAX-MIN BCP, which is the first constant approximation for this problem.

► **Theorem 11.** *A 3-approximation for the MAX-MIN BCP problem can be computed in $\mathcal{O}(\log(X^*)k^2|V||E|)$, where X^* denotes the optimal value.*

Proof. Let (G, k) be an instance of MAX-MIN BCP. For any value X , using BCD, we show how to either obtain an $[X/3, \infty)$ -CVP $_k$, or report that $X > X^*$. Once we have this procedure in hand, a binary search can be used to obtain an $[X^*/3, \infty)$ -CVP $_k$.

We first obtain a λ -BCD (C, H, \mathfrak{R}, f) of G with $\lambda = \lceil X/3 \rceil$. If $|H| + |\mathfrak{R}| \geq k$, we output a $[X/3, \infty)$ -CVP $_k$ given by the body and the assignment to head vertices (if this gives more than k sets, arbitrarily merge some until there are only k). If $|H| + |\mathfrak{R}| < k$, then we report that $X > X^*$. To see that this is correct, assume towards contradiction that $X \leq X^*$, and consider an optimal solution $\mathcal{S}^* = \{S_1^*, \dots, S_k^*\}$. Then in the λ -BCD we computed, we know that $w(R) < X$ for every $R \in \mathfrak{R}$ and $w(C') < X$ for every $C' \in \mathcal{CC}(C)$. Observe that then no $C' \in \mathcal{CC}(C)$ or a subset of it can be a set in \mathcal{S}^* , since $w(S_i^*) \geq X^* \geq X$ for every $S_i^* \in \mathcal{S}^*$. From the separator properties of H and that the fact that each $S_i^* \in \mathcal{S}^*$ is connected, we obtain that any set in \mathcal{S}^* containing vertices from C also has to contain at least one vertex from H . Thus, we can derive that the cardinality of $\mathcal{S}_H^* = \{S_i^* \in \mathcal{S}^* \mid S_i^* \cap (C \cup H) \neq \emptyset\}$ is at most $|H|$. Also, $|\mathcal{S}^* \setminus \mathcal{S}_H^*| \leq w(V(\mathfrak{R}))/X^* \leq w(V(\mathfrak{R}))/X \leq |\mathfrak{R}|$. Thus it follows that $|\mathcal{S}^*| \leq |H| + |\mathfrak{R}| < k$, a contradiction. ◀

The last problem that we consider as application of the balanced crown decomposition is the MIN-MAX BCP problem, where we also provide the first constant approximation result.

► **Theorem 12.** *A 3-approximation for the MIN-MAX BCP problem can be computed in time $\mathcal{O}(\log(X^*)|V||E|(\log \log X^* \log(|V|w_{max}) + k^2))$, where X^* denotes the optimum value and $w_{max} = \max_{v \in V} w(v)$ the maximum weight of a vertex.*

Proof. Achieving this requires several technical steps after having a balanced crown decomposition in hand, including a second use of the balanced expansion. Let (G, k) be an instance of MIN-MAX BCP and let $\mathcal{S}^* = \{S_1^*, \dots, S_k^*\}$ be an optimal solution. Let (C, H, \mathfrak{R}, f) be a λ -BCD of G . Similar to the Max-Min case we try to make a comparison between \mathcal{S}^* and the vertex decomposition C, H and $V(\mathfrak{R})$. The main issue is that, in contrast to the Max-Min case, an optimal solution can (and sometimes has to) build more than $|H|$ sets from the vertices in $H \cup C$. With the connectivity constraints, this means that some components in $G[C]$ are in fact a set in the optimal partition. Hence, when computing an approximate solution from a balanced crown decomposition, we have to also choose some components from $G[C]$ to be sets, while others are combined with some vertex in H . In order to make the decision of where to place the components in $G[C]$, we compute a min-cost flow using the algorithm from [1] on a network that models the options for components in $G[C]$ to either be sets or be combined with some vertex in H . A partial embedding of $\{S_i^* \in \mathcal{S}^* \mid S_i^* \cap (C \cup H) \neq \emptyset\}$ to this cost-flow network allows a comparison with the resulting

partition of $C \cup H$. The balanced weight properties of \mathfrak{R} then yield a comparison with the whole set S^* . With the additional use of a min cost-flow network, our balanced crown structure can be used to estimate the optimal objective value and again enables a binary search for an approximate solution. ◀


References

- 1 Ravindra K Ahuja, Andrew V Goldberg, James B Orlin, and Robert E Tarjan. Finding minimum-cost flows by double scaling. *Mathematical programming*, 53(1-3):243–266, 1992.
- 2 Curtis A Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs a comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1:13–22, 1998.
- 3 Ralf Borndörfer, Ziena Elijazyfer, and Stephan Schwartz. Approximating balanced graph partitions. Technical Report 19-25, ZIB, Takustr. 7, 14195 Berlin, 2019.
- 4 Christoph Brause and Ingo Schiermeyer. Kernelization of the 3-path vertex cover problem. *Discrete Mathematics*, 339(7):1935–1939, 2016. doi:10.1016/j.disc.2015.12.006.
- 5 Paolo M Camerini, Giulia Galbiati, and Francesco Maffioli. On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, 5(1):39–50, 1983.
- 6 L. Sunil Chandran, Yun Kuen Cheung, and Davis Issac. Spanning tree congestion and computation of generalized györi-lovász partition. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 32:1–32:14, 2018.
- 7 Frédéric Chataigner, Liliane Benning Salgado, and Yoshiko Wakabayashi. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics and Theoretical Computer Science*, 9(1), 2007. URL: <http://dmtcs.episciences.org/384>.
- 8 Guangting Chen, Yong Chen, Zhi-Zhong Chen, Guohui Lin, Tian Liu, and An Zhang. Approximation algorithms for the maximally balanced connected graph tripartition problem. *Journal of Combinatorial Optimization*, pages 1–21, 2020.
- 9 Jianer Chen, Henning Fernau, Peter Shaw, Jianxin Wang, and Zhibiao Yang. Kernels for packing and covering problems. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 199–211. Springer, 2012.
- 10 Jiangzhuo Chen, Robert D Kleinberg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (almost) tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM (JACM)*, 54(4):16, 2007.
- 11 Yong Chen, Zhi-Zhong Chen, Guohui Lin, Yao Xu, and An Zhang. Approximation algorithms for maximally balanced connected graph partition. In *International Conference on Combinatorial Optimization and Applications*, pages 130–141. Springer, 2019.
- 12 Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225–230, 1996.
- 13 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $o(n^2)$ steps. In *Graph-Theoretic Concepts in Computer Science, 30th International Workshop*, volume 3353 of *LNCS*, pages 257–269. Springer, 2004.
- 14 Pål Grønås Drange, Markus S. Dregi, and Pim van ’t Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.
- 15 Rosa Enciso, Michael R Fellows, Jiong Guo, Iyad Kanj, Frances Rosamond, and Ondřej Suchý. What makes equitable connected partition easy. In *International Workshop on Parameterized and Exact Computation*, pages 122–133. Springer, 2009.
- 16 Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM Journal on Discrete Mathematics*, 30(1):383–410, 2016.

- 18 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 19 Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 168–177. ACM/SIAM, 1991. URL: <http://dl.acm.org/citation.cfm?id=127787.127822>.
- 20 E Gyori. On division of graphs to connected subgraphs, combinatorics. In *Colloq. Math. Soc. Janos Bolyai, 1976*, 1976.
- 21 Alexander Hoyer. *On the Independent Spanning Tree Conjectures and Related Problems*. PhD thesis, Georgia Institute of Technology, 2019.
- 22 Jörg Kalcsics and Roger Z. Ríos-Mercado. *Districting Problems*, pages 705–743. Springer International Publishing, Cham, 2019.
- 23 Atsushi Kaneko, Alexander K. Kelmans, and Tsuyoshi Nishimura. On packing 3-vertex paths in a graph. *Journal of Graph Theory*, 36(4):175–197, 2001. doi:10.1002/1097-0118(200104)36:4%3C175::AID-JGT1005%3E3.0.CO;2-T.
- 24 Mithilesh Kumar and Daniel Lokshtanov. A 2lk kernel for l-component order connectivity. In *11th International Symposium on Parameterized and Exact Computation*, volume 63 of *LIPIcs*, pages 20:1–20:14, 2016. doi:10.4230/LIPIcs.IPEC.2016.20.
- 25 Sukhamay Kundu and Jayadev Misra. A linear tree partitioning algorithm. *SIAM Journal on Computing*, 6(1):151–154, 1977.
- 26 Wenjun Li, Junjie Ye, and Yixin Cao. Kernelization for p_2 -packing: A gerrymandering approach. In *Frontiers in Algorithmics - 12th International Workshop*, volume 10823 of *LNCS*, pages 140–153. Springer, 2018. doi:10.1007/978-3-319-78455-7_11.
- 27 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, 36:177–189, 1979.
- 28 László Lovász. A homology theory for spanning trees of a graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 30(3-4):241–251, 1977.
- 29 James B Orlin. Max flows in $o(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774, 2013.
- 30 Yehoshua Perl and Stephen R Schach. Max-min tree partitioning. *Journal of the ACM (JACM)*, 28(1):5–15, 1981.
- 31 Arnold L. Rosenberg and Lenwood S. Heath. *Graph separators with applications*. Frontiers of computer science. Springer, 2001.
- 32 Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Information Processing Letters*, 33(5):227–231, 1990.
- 33 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- 34 Jianhua Tu, Lidong Wu, Jing Yuan, and Lei Cui. On the vertex cover p_3 problem parameterized by treewidth. *Journal of Combinatorial Optimization*, 34(2):414–425, 2017. doi:10.1007/s10878-016-9999-6.
- 35 Koichi Wada and Kimio Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 132–143. Springer, 1993.
- 36 Jianxin Wang, Dan Ning, Qilong Feng, and Jianer Chen. An improved kernelization for p_2 -packing. *Information Processing Letters*, 110(5):188–192, 2010. doi:10.1016/j.ipl.2009.12.002.
- 37 Lele Wang, Zhao Zhang, Di Wu, Weili Wu, and Lidan Fan. Max-min weight balanced connected partition. *Journal of Global Optimization*, 57(4):1263–1275, 2013.
- 38 Bang Ye Wu. A $7/6$ -approximation algorithm for the max-min connected bipartition problem on grid graphs. In *International Conference on Computational Geometry, Graphs and Applications*, pages 188–194. Springer, 2010.

- 39 Bang Ye Wu. Fully polynomial-time approximation schemes for the max–min connected partition problem on interval graphs. *Discrete Mathematics, Algorithms and Applications*, 4(1), 2012.
- 40 Mingyu Xiao. Linear kernels for separating a graph into components of bounded size. *Journal of Computer and System Sciences*, 88:260–270, 2017.
- 41 Mingyu Xiao and Shaowei Kou. Kernelization and parameterized algorithms for 3-path vertex cover. In *Theory and Applications of Models of Computation*, volume 10185 of *LNCS*, pages 654–668, 2017. doi:10.1007/978-3-319-55911-7_47.
- 42 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. doi:10.1137/0210022.
- 43 Xing Zhou, Huaimin Wang, Bo Ding, Tianjiang Hu, and Suning Shang. Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications*, 116:10–20, 2019.

All-Pairs Shortest Paths for Real-Weighted Undirected Graphs with Small Additive Error

Timothy M. Chan 

Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract

Given a graph with n vertices and *real* edge weights in $[0, 1]$, we investigate an approximate version of the standard all-pairs shortest paths (APSP) problem where distances are estimated with *additive error* at most ε . Yuster (2012) introduced this natural variant of approximate APSP, and presented an algorithm for directed graphs running in $\tilde{O}(n^{(3+\omega)/2}) \leq O(n^{2.687})$ time for an arbitrarily small constant $\varepsilon > 0$, where ω denotes the matrix multiplication exponent. We give a faster algorithm for *undirected* graphs running in $\tilde{O}(n^{(3+\omega^2)/(\omega+1)}) \leq O(n^{2.559})$ time for any constant $\varepsilon > 0$. If $\omega = 2$, the time bound is $\tilde{O}(n^{7/3})$, matching a previous result for undirected graphs by Dor, Halperin, and Zwick (2000) which only guaranteed additive error at most 2.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Shortest paths, approximation, matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.27

1 Introduction

The *all-pairs shortest paths* (APSP) problem is one of the most well-known problems in algorithm design, and plays a central role in the study of fine-grained complexity. Only small (subpolynomial) speedups over the textbook cubic-time algorithms are known for arbitrary real-weighted dense graphs [6, 8, 13, 21], and it has been conjectured that there are no truly subcubic algorithms [20]. If faster running time is desired, one thus turns to *approximation algorithms*.

To this end, Zwick [23] described an $O(n^\omega \log \frac{w_{\max}}{w_{\min}})$ -time algorithm with approximation factor $1 + \varepsilon$ for any constant $\varepsilon > 0$, for any (directed or undirected) graph with positive real edge weights, where n is the number of vertices, w_{\min} and w_{\max} denote the minimum and maximum edge weight, and $\omega < 2.373$ is the matrix multiplication exponent [2, 11]. For every pair of vertices u and v , the algorithm computes a value $\tilde{D}[u, v]$ such that $D[u, v] \leq \tilde{D}[u, v] \leq (1 + \varepsilon)D[u, v]$, where $D[u, v]$ denotes the distance (i.e., the shortest-path weight) from u to v .

While multiplicative approximation is natural, in this paper we are interested in an even stronger form of approximation, where we want small *additive error* bounded by εw_{\max} . More precisely, for every u and v , we seek a value $\tilde{D}[u, v]$ such that $D[u, v] \leq \tilde{D}[u, v] \leq D[u, v] + \varepsilon w_{\max}$. To see how this can yield a much better estimate, imagine the case when the distance $D[u, v]$ is large; a $(1 + \varepsilon)$ -factor approximation may differ from the true value by $\varepsilon D[u, v]$, which could be much bigger than ε times the maximum weight of a single edge.¹

From now on, we assume $w_{\max} = 1$, without loss of generality, by rescaling. In other words, we assume that all edge weights lie in $[0, 1]$ and we tolerate additive error at most ε .

¹ In fact, we can achieve additive error at most $O(\varepsilon w_{\max}[u, v])$, where $w_{\max}[u, v]$ denotes the weight of the longest edge in a shortest path from u to v , by guessing a value $w \in [w_{\max}[u, v]/2, w_{\max}[u, v]]$ and removing all edges of weight exceeding w from the graph; $O(\log \frac{w_{\max}}{w_{\min}})$ guesses of w suffice.



© Timothy M. Chan;

licensed under Creative Commons License CC-BY 4.0

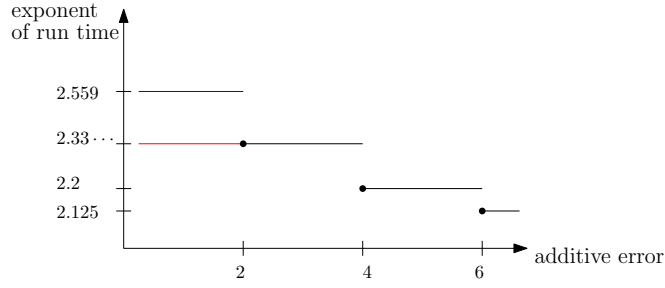
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 27; pp. 27:1–27:9

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Tradeoffs between additive error and running time for approximate APSP in undirected real-weighted graphs. The blue part indicates the new result using the current matrix multiplication bound $\omega < 2.373$; the red part indicates the new result if $\omega = 2$.

This form of additive approximation was first considered by Yuster [22], who gave an algorithm running in $\tilde{O}(n^{(3+\omega)/2}) \leq O(n^{2.687})$ time² for any constant additive error $\varepsilon > 0$. If $\omega = 2$, the time bound is $\tilde{O}(n^{2.5})$. For arbitrary (dense) directed graphs, this would be difficult to improve, since approximate APSP with additive error at most c for any constant c is at least as hard as exact unweighted APSP, for which the current best algorithm by Zwick [23] has exponent 2.5 when $\omega = 2$ and is conjectured to be near optimal [9]. However, a question remains as to whether an improved result for approximate APSP with additive error ε is possible for *undirected* graphs (for which $\tilde{O}(n^\omega)$ -time algorithms are known for exact unweighted APSP [15, 19]).

For *unweighted undirected* graphs, the seminal work by Aingworth, Chekuri, Indyk, and Motwani [1] described combinatorial algorithms for approximate APSP achieving $O(1)$ additive error without using fast matrix multiplication. Subsequent improvements were described by Dor, Halperin, and Zwick [12]: in particular, for dense graphs, the best results were an $\tilde{O}(n^{7/3})$ -time algorithm with additive error at most 2, and an $\tilde{O}(n^{2+2/(3k-2)})$ -time algorithm with additive error at most k for any even constant $k > 2$. As noted in Dor et al.'s paper, these algorithms for unweighted undirected graphs can actually be extended to weighted undirected graphs with arbitrary real edge weights in $[0, 1]$. However, none of these results achieve additive error below 2. In fact, approximate undirected APSP with additive error strictly below 2 is at least as hard as Boolean matrix multiplication (by considering tripartite graphs with unit edge weights), thus ruling out combinatorial algorithms for arbitrarily small ε .

New result. Our main result is a new algorithm for undirected real-weighted graphs with additive error ε . The algorithm uses fast matrix multiplication and achieves running time $\tilde{O}(n^{(3+\omega^2)/(\omega+1)}) \leq O(n^{2.559})$. This improves Yuster's $O(n^{2.689})$ -time directed-graph algorithm. Furthermore, if $\omega = 2$, our time bound becomes $\tilde{O}(n^{7/3})$, which improves Yuster's $\tilde{O}(n^{2.5})$ bound and also matches Dor, Halperin, and Zwick's time bound for additive error 2 while making the additive error an arbitrarily small constant ε . (See Figure 1.)

Other related work. Other formulations of approximate APSP have been studied in the literature. For example, Bringmann, Künnemann, and Węgrzycki [5] revisited multiplicative approximation, but in a setting where $\frac{w_{\max}}{w_{\min}}$ may be large (Zwick's approximation algorithm [23] works well only when this ratio is bounded). On the other hand, Roditty and

² The \tilde{O} notation hides $\log^{O(1)} n$ factors. Factors dependent on ε are also suppressed, for simplicity, but they will all be polynomial in $1/\varepsilon$.

Shapira [18] gave results (recently improved by Chan, Vassilevska Williams, and Xu [9]) on approximate APSP for unweighted directed APSP that achieved *sublinear* additive error bounded by $D[u, v]^p$ for a given constant $p < 1$; the guarantee is stronger than $1 + \varepsilon$ multiplicative approximation as $D[u, v]$ gets larger, but is not as strong as constant additive error.

In the literature on graph spanners, there have also been many results for unweighted undirected graphs with constant additive errors [1, 4, 10].

Our time bound appears unusual among algorithms in the literature that rely on fast matrix multiplication. Coincidentally, Grandoni et al. [16] gave a matrix-multiplication-based algorithm for an entirely different problem (all-pairs lowest common ancestors in DAGs) with a time bound that is also $\tilde{O}(n^{7/3})$ if $\omega = 2$ (though their exponent got below 2.5 under the current matrix multiplication bound). For our result, $n^{7/3}$ is a natural barrier, since even with additive error 2, the current best algorithm by Dor, Halperin, and Zwick [12] still requires $\tilde{O}(n^{7/3})$ time.

Techniques. To see why the additive approximation problem is challenging for real-weighted graphs, consider the standard idea of rounding edge weights and rescaling to turn them into integers. To guarantee additive error at most a constant ε , since a shortest path may require $\Theta(n)$ edges in the worst case, one would need to round edge weights to multiples of ε/n , and the rescaled integers could then be $\Theta(n)$; unfortunately, APSP (or min-plus product) for integer input of that magnitude still requires near cubic time under the current state of the art.

Yuster’s previous algorithm for directed graphs [22] is based on an approach by Zwick [23] (originally for exact small-edge-weighted APSP), which divides into two cases: paths that use a few edges (i.e., hops) vs. paths that use many edges. For shortest paths with less than L edges, the aforementioned rounding approach reduces the problem to computing min-plus products for small integers bounded by $O(L)$, which reduces to standard matrix multiplication on $\tilde{O}(L)$ -bit numbers. On the other hand, for shortest paths with more than L edges, there exists a small hitting set (also called a “bridging set”) with $\tilde{O}(n/L)$ vertices, and we can run a single-source/single-sink algorithm from/to each such vertex. Finally, the parameter L is chosen (near \sqrt{n} if $\omega = 2$) to balance cost.

To improve the running time, we combine this approach with Aingworth et al.’s approach [1] (which Dor et al.’s algorithm [12] builds upon). Aingworth et al.’s approach divides into two cases differently: vertices with low degree vs. vertices with high degree. Low-degree vertices are less expensive because the number of incident edges is small. On the other hand, for high-degree vertices, there exists a small dominating set, and so these vertices can be covered by a small number of “clusters”; sources in the same cluster are close together, and so distances from one fixed source s give us a good approximation (with $O(1)$ additive error) to distances from other sources in the same cluster (since the graph is undirected). To reduce the additive error from $O(1)$ to $O(\varepsilon)$, we need a number of further ideas. We will use min-plus products at each layer of the BFS tree from s . To make the error sum to $O(\varepsilon)$, we will set the error tolerance at each layer to be proportional to the size of the layer (roughly $\varepsilon n_i/n$ if the i -th layer has size n_i , as we will describe in Section 4). Bounding the total running time requires some care, since the min-plus products are done to matrices of different dimensions at the different layers.

The general plan shares some similarity with an exact combinatorial APSP algorithm by Chan [7] for sparse undirected unweighted graphs, which also modifies Aingworth et al.’s algorithm and simulates BFS to compute distances from multiple sources in a cluster.

However, what makes our algorithm interesting is the combination of Aingworth et al.'s approach with fast matrix multiplication. (A recent algorithm by Chan, Vassilevska Williams, and Xu [9] for a different problem – all-pairs lightest shortest paths for undirected small-weighted graphs – also combines Aingworth et al.'s approach with matrix multiplication, but does not involve approximation nor real weights. Our algorithm here is more elaborate.)

2 Preliminaries

Given two matrices A and B , we let $A \star B$ denote the min-plus product, i.e., $(A \star B)[u, v] = \min_z (A[u, z] + B[z, v])$.

Let $\mathcal{M}(n_1, n_2, n_3)$ denote the time complexity for computing the standard product of an $n_1 \times n_2$ and an $n_2 \times n_3$ matrix.

Let $\mathcal{M}^*(n_1, n_2, n_3 \mid \ell)$ denote the time complexity for computing the min-plus product of an $n_1 \times n_2$ and an $n_2 \times n_3$ matrix, where all the matrix entries are from $\{0, 1, \dots, \ell, \infty\}$. As is well known [3], $\mathcal{M}^*(n_1, n_2, n_3 \mid \ell) \leq \tilde{O}(\ell \cdot \mathcal{M}(n_1, n_2, n_3))$.

3 Small Distances

We begin with a lemma on computing small distances, which will be useful later. Yuster [22] has already observed how to solve the problem for paths with small number of edges, but our lemma is more challenging, since a path with small weight could still have a large number of hops.

► **Lemma 1.** *Given a directed or undirected graph $G = (V, E)$ with n vertices and real edge weights in $[0, 1]$, and given $\beta > \varepsilon$, we can approximate all distances that are at most β , with additive error $O(\varepsilon)$, in $\tilde{O}((\beta/\varepsilon)n^\omega)$ time.*

Proof. We use a form of repeated squaring: loosely speaking, we recursively solve the problem for $\beta/2$ and compute the min-plus product of the resulting matrix with itself. The additive error ε needs to be roughly halved in the recursive call, but luckily the ratio β/ε stays roughly the same.

Let $\delta > 0$ be a parameter to be set later. Let β_{\min} be the smallest edge weight. Assume that $\beta > \varepsilon$. For every $u, v \in V$, we will compute $\tilde{D}^{(\beta, \varepsilon)}[u, v]$, an approximation to $D[u, v]$ with additive error at most ε , provided that $D[u, v] \leq \beta$. (More precisely, if $\tilde{D}^{(\beta, \varepsilon)}[u, v] \neq \infty$, it is a valid approximation; and if $D[u, v] \leq \beta$, then it is guaranteed that $\tilde{D}^{(\beta, \varepsilon)}[u, v] \neq \infty$.)

To compute $\tilde{D}^{(\beta, \varepsilon)}$:

1. First recursively compute $D' = \tilde{D}^{(\beta/2, (1-\delta)\varepsilon/2)}$. Round the entries in D' (upward) to multiples of $\delta\varepsilon/3$.
2. Let $A'[u, v]$ be $w(u, v)$ rounded (upward) to a multiple of $\delta\varepsilon/3$. If $A'[u, v] > \beta$, reset $A'[u, v] = \infty$.
3. Set $\tilde{D}^{(\beta, \varepsilon)} = D' \star A' \star D'$. If $\tilde{D}^{(\beta, \varepsilon)}[u, v] > \beta + \varepsilon$, reset $\tilde{D}^{(\beta, \varepsilon)}[u, v] = \infty$.

Correctness follows since any path with weight at most β can be expressed as $\pi_1 e \pi_2$, where each of π_1 and π_2 is a subpath with weight at most $\beta/2$, and e is a single edge (the “median”). The additive error is bounded by $2(1-\delta)\varepsilon/2 + \delta\varepsilon/3 + \delta\varepsilon/3 + \delta\varepsilon/3 = \varepsilon$.

Since the finite entries of D' and A' after rescaling are integers bounded by $O(\frac{\beta}{\delta\varepsilon})$, these min-plus products take $O(\mathcal{M}^*(n, n, n \mid \frac{\beta}{\delta\varepsilon})) = \tilde{O}(\frac{\beta}{\delta\varepsilon} n^\omega)$ time. The total time satisfies the recurrence

$$T(\beta, \varepsilon) = T(\beta/2, (1-\delta)\varepsilon/2) + \tilde{O}(\frac{\beta}{\delta\varepsilon} n^\omega),$$

which yields $T(\beta, \varepsilon) = \tilde{O}(\frac{\beta}{\delta\varepsilon} (\frac{1}{1-\delta})^{\log(\beta/\beta_{\min})} n^\omega)$.

We may assume that $\beta \leq n$ and $\beta_{\min} \geq \varepsilon/n$, since we can initially round edge weights to multiples of ε/n . We set $\delta = 1/\log(\beta/\beta_{\min}) = \Omega(1/\log(n/\varepsilon))$, so that $(\frac{1}{1-\delta})^{\log(\beta/\beta_{\min})}$ is bounded by a constant. \blacktriangleleft

4 Multiple Sources

Next, we solve the subproblem of approximating shortest paths from multiple sources in a close-knit “cluster” of vertices $S \subseteq V$. (To optimize the final running time, we find it useful to separate out a preprocessing stage that does not depend on S .)

► Lemma 2. *Given an undirected graph $G = (V, E)$ with n vertices and real edge weights in $[0, 1] \cup \{\infty\}$, and a parameter B , we can preprocess in $\tilde{O}((1/\varepsilon)Bn^\omega)$ time, so that: given a subset $S \subseteq V$ of size $O(n/t)$ where every pair of vertices in S have distance at most an integer constant c , we can approximate the distances for all pairs in $S \times V$ with additive error $O(\varepsilon)$, in $\tilde{O}((1/\varepsilon)(n^\omega/B^{\omega-2} + n^\omega/t^{\omega-2}))$ time.*

Proof. Let $\varepsilon' = \varepsilon/\log n$. Fix a vertex $s \in S$. We first compute all distances from s in $O(n^2)$ time by Dijkstra’s algorithm. Let $V_i = \{v \in V : D[s, v] \in [i, i+1)\}$. For any interval I , let $V_I = \bigcup_{i \in I} V_i$. Let $n_i = |V_{[i-2c-3, i+2c+3]}|$; note that $\sum_i n_i = O(n)$.

We describe an algorithm to compute a partial matrix \tilde{D} of approximate distances. For subsets $S_1, S_2 \subseteq V$, let $\tilde{D}(S_1, S_2)$ denote the submatrix of \tilde{D} containing only entries for $(u, v) \in S_1 \times S_2$.

Step 0. For each i , we compute $\tilde{D}(V_{i-1}, V_i)$ by applying Lemma 1 to approximate distances between all $u \in V_{i-1}$ and all $v \in V_i$ that are bounded by $2c+1$, in the subgraph induced by $V_{[i-2c-3, i+2c+3]}$ (which has size $O(n_i)$), with additive error $O(\frac{\varepsilon' n_i}{n})$. (More precisely, for $u \in V_{i-1}$ and $v \in V_i$, if $\tilde{D}[u, v] \neq \infty$, then the computed value $\tilde{D}[u, v]$ is a valid approximation; and if u and v have distance at most $2c+1$ in the induced subgraph, then it is guaranteed that $\tilde{D}[u, v] \neq \infty$.) For all i with $n_i \leq n/B$, the total running time is

$$\begin{aligned} \tilde{O}\left(\sum_i \frac{n}{\varepsilon' n_i} \cdot n_i^\omega\right) &= \tilde{O}\left(\sum_i \frac{n}{\varepsilon'} \cdot n_i^{\omega-1}\right) \\ &\leq \tilde{O}\left(\sum_i \frac{n}{\varepsilon'} \cdot (n/B)^{\omega-2} \cdot n_i\right) \\ &= \tilde{O}\left(\frac{n}{\varepsilon'} \cdot (n/B)^{\omega-2} \cdot n\right) = \tilde{O}((1/\varepsilon')n^\omega/B^{\omega-2}). \end{aligned}$$

In the case when $n_i > n/B$, we apply Lemma 1 instead to the original graph with additive error $O(\varepsilon'/B)$, which is at least as good as $O(\frac{\varepsilon' n_i}{n})$, in $\tilde{O}((1/\varepsilon')Bn^\omega)$ time – note that this can be done just once during preprocessing.

Step 1. For $i = 0, \dots, c$, we compute $\tilde{D}(S, V_i)$ by using Lemma 1 to approximate all distances bounded by $O(c)$, with additive error $O(\varepsilon')$.

For each $i = c+1, \dots, t$, we compute $\tilde{D}(S, V_i)$ by taking the min-plus product $\tilde{D}(S, V_{i-1}) \star \tilde{D}(V_{i-1}, V_i)$, with additive error $O(\frac{\varepsilon' n_i}{n})$. We do the following *filtering* step: for each entry $\tilde{D}[x, y]$ just computed, if $\tilde{D}[x, y] \notin D[s, y] - D[s, x] \pm (2c + O(\varepsilon))$, reset $\tilde{D}[x, y] = \infty$. Because of the filtering step, for all $u \in S$, $z \in V_{i-1}$, and $v \in V_i$, we have $\tilde{D}[u, z] \in i \pm O(c)$ if it is finite, and $\tilde{D}[z, v] \in O(c)$ if it is finite. Thus, in computing $\tilde{D}(S, V_{i-1}) \star \tilde{D}(V_{i-1}, V_i)$, we can

make the matrix entries lie in $O(c)$ by shifting. We can then round entries to multiples of $\frac{n}{\varepsilon' n_i}$. So, the total time to compute the products for all $i = c + 1, \dots, t$ is

$$\begin{aligned}
\tilde{O}\left(\sum_{i=1}^t \mathcal{M}^*(n/t, n_i, n_i \mid \frac{n}{\varepsilon' n_i})\right) &\leq \tilde{O}\left(\sum_{i=1}^t \frac{n}{\varepsilon' n_i} \cdot \mathcal{M}(n/t, n_i, n_i)\right) \\
&\leq \tilde{O}\left(\sum_{i=1}^t \frac{n}{\varepsilon' n_i} \cdot \left(\frac{n}{n_i} n_i^\omega + \left(\frac{n_i}{n/t}\right)^2 (n/t)^\omega\right)\right) \\
&= \tilde{O}\left(\frac{1}{\varepsilon'} \sum_{i=1}^t ((n^2/t) n_i^{\omega-2} + (n^{\omega-1}/t^{\omega-2}) n_i)\right) \\
&\leq \tilde{O}\left(\frac{1}{\varepsilon'} \cdot ((n^2/t) n^{\omega-2} t^{3-\omega} + (n^{\omega-1}/t^{\omega-2}) n)\right) \\
&= \tilde{O}((1/\varepsilon') n^\omega / t^{\omega-2}).
\end{aligned}$$

Step 2. Now, assume that $\tilde{D}(S, V_i)$ has been computed for all $i = 0, \dots, \ell/2$ for a given $\ell \geq 2t$. We will compute $\tilde{D}(S, V_i)$ for all $i = \ell/2 + 1, \dots, \ell$. First pick an $i_0 \leq \ell/2$ with $|V_{i_0}| = O(n/\ell)$. For each $i = i_0 + 1, \dots, \ell$, we compute $\tilde{D}(V_{i_0}, V_i)$ by taking the min-plus product $\tilde{D}(V_{i_0}, V_{i-1}) \star \tilde{D}(V_{i-1}, V_i)$, with additive error $O(\frac{\varepsilon' n_i}{n})$. Do the filtering step as before. Because of the filtering step, for all $u \in V_{i_0}$, $z \in V_{i-1}$, and $v \in V_i$, we have $\tilde{D}[u, z] \in i - i_0 \pm O(c)$ if it is finite, and $\tilde{D}[z, v] \in O(c)$ if it is finite. By a similar analysis, the total time to compute these products for $i = i_0 + 1, \dots, \ell$ is upper-bounded by

$$\tilde{O}\left(\sum_{i=1}^{\ell} \mathcal{M}^*(n/\ell, n_i, n_i \mid \frac{n}{\varepsilon' n_i})\right) \leq \tilde{O}((1/\varepsilon') n^\omega / \ell^{\omega-2}) \leq \tilde{O}((1/\varepsilon') n^\omega / t^{\omega-2}).$$

Finally, we compute $\tilde{D}(S, V_{(\ell/2, \ell]})$ by taking the min-plus product $\tilde{D}(S, V_{i_0})$ and $\tilde{D}(V_{i_0}, V_{(\ell/2, \ell]})$ with additive error $O(\varepsilon')$. Do the filtering step as before. Because of the filtering step, for all $u \in S$, $z \in V_{i_0}$, and $v \in V_{(\ell/2, \ell]}$, we have $\tilde{D}[u, z] \in i_0 \pm O(c)$ if it is finite, and $\tilde{D}[z, v] \in D[s, v] - i_0 \pm O(c)$ if it is finite. We can again make the matrix entries lie in $O(c)$ by shifting. This product takes time

$$\begin{aligned}
\tilde{O}(\mathcal{M}^*(n/t, n/\ell, n \mid 1/\varepsilon')) &\leq \tilde{O}((1/\varepsilon') \mathcal{M}(n/t, n/t, n)) \\
&\leq \tilde{O}((1/\varepsilon') t (n/t)^\omega) = \tilde{O}((1/\varepsilon') n^\omega / t^{\omega-1}).
\end{aligned}$$

We repeat the above for all $\ell \geq 2t$ that are powers of 2.

Correctness. For every $u \in S$ and $v \in V$, we claim that $\tilde{D}[u, v]$ approximates $D[u, v]$ with additive error $O(\varepsilon)$. To see this, let π be the shortest path from u to v . For any subpath of π , say, from x to y , we have $D[x, y] = D[u, y] - D[u, x] \in D[s, y] - D[s, x] \pm 2c$, since $D[s, u] \leq c$ and the graph is undirected (this justifies the filtering step). If $u \in S$ and $v \in V_i$ with $i > c$, then π must pass through a vertex $z \in V_{i-1}$. For every node z' in the subpath from z to v , $D[z, z'] \leq D[u, v] - D[u, z] \leq D[s, v] - D[s, z] + 2c \leq 2c + 2$, and $D[s, z'] \in D[s, z] \pm D[z, z'] \in i \pm (2c + 3)$, so the subpath lies in the subgraph induced by $V_{[i-2c-3, i+2c+3]}$. It follows that the total additive error in Step 1 is $O(\sum_i \frac{\varepsilon' n_i}{n}) = O(\varepsilon')$. The analysis of Step 2 is similar: if $u \in S$ and $v \in V_i$ with $i > \ell/2$, then π must pass through a vertex $u' \in V_{i_0}$, and the subpath from u' to v must pass through a vertex $z \in V_{i-1}$. The overall additive error is bounded by $O(\varepsilon' \log n) = O(\varepsilon)$. ◀

5 Overall Algorithm

Our overall algorithm employs a careful combination of Aingworth et al.'s technique [1] involving low- vs. high-degree vertices, and Zwick's technique [23] involving short vs. long paths, and a judicious choice of several parameters to balance cost.

Let B , L , and t be parameters to be set later. Let V_{high} be the set of all vertices of degree more than n/t , and V_{low} be the set of all vertices of degree at most n/t .

Phase 1. We will first compute an approximation $\tilde{D}[u, v]$ to $D[u, v]$ for all $u \in V_{\text{high}}$ and $v \in V$, as follows:

Let $X \subseteq V$ be a dominating set for V_{high} , such that every vertex in V_{high} is in the (closed) neighborhood of some vertex in X . As noted by Aingworth et al. [1], there exists such a dominating set of size $\tilde{O}(t)$, and it can be found easily by random sampling, or deterministically by a greedy algorithm.

Consequently, we can cover V_{high} by $\tilde{O}(t)$ groups of vertices, such that vertices of each group have distance at most 2 from each other, by taking the neighborhoods of the vertices of X . We may assume that these groups are disjoint (for example, by removing from the i -th group those vertices that appear in the first $i - 1$ groups). Furthermore, we may assume that every group has size $O(n/t)$ (by subdividing the groups, which only increases the number of groups by $O(t)$). For each group S , we apply Lemma 2 (with $c = 2$). The total time of these $\tilde{O}(t)$ invocations of the lemma is $\tilde{O}(t \cdot (1/\varepsilon)n^\omega/B^{\omega-2})$, assuming that $B \leq t$, after $\tilde{O}((1/\varepsilon)Bn^\omega)$ -time preprocessing.

Phase 2. Let $R \subseteq V$ be a subset of vertices that hits all shortest paths with at least L edges. As shown by Zwick [23], there exists such a hitting set of size $\tilde{O}(n/L)$, and it can be found by random sampling, or deterministically. We will next compute an approximation $\hat{D}[u, v]$ to $D[u, v]$ for all $u \in R$ and $v \in V$ as follows:

Fix $u \in R$. Define a graph G_u containing all edges xy with $x \in V_{\text{low}}$ or $y \in V_{\text{low}}$; for each $z \in V_{\text{high}}$, we add an extra edge uz with weight $\tilde{D}[u, z]$, which has been computed in Phase 1. Then the distance from u to v in G_u approximates the distance in G (because if $\langle u_1, \dots, u_k \rangle$ is a shortest path in G with $u_1 = u$, and i is the largest index with $u_i \in V_{\text{high}}$, then $\langle u_1, u_i, \dots, u_k \rangle$ is a path in G_u). We run Dijkstra's algorithm on G_u from the source u . Since G_u has $O(n^2/t)$ edges, this takes $\tilde{O}(n^2/t)$ time per u . The total over all $u \in R$ is $\tilde{O}((n/L) \cdot (n^2/t)) = \tilde{O}(n^3/(tL))$.

Phase 3. We now approximate $D[u, v]$ for all $u, v \in V$ as follows.

For (u, v) with $D[u, v] \leq L$, we use Lemma 1, which takes $\tilde{O}((1/\varepsilon)Ln^\omega)$ time.

For (u, v) with $D[u, v] > L$, recall that we have computed $\hat{D}(V, R)$ from Phase 2. For $u \in V$ and $z \in R$, let $D'[u, z] = \hat{D}[u, z]$ if $\hat{D}[u, z] \leq L + O(\varepsilon)$, and $D'[u, z] = \infty$ otherwise. For $z \in R$ and $v \in V$, let $D''_a[z, v] = (\hat{D}[z, v] + a) \bmod 10L$. Compute the min-plus product $D'(V, R) \star D''_a(R, V)$ with additive error $O(\varepsilon)$ for $a = 0$ and for $a = 5L$. Since the finite entries after rescaling are integers bounded by $O(L/\varepsilon)$, this takes $\tilde{O}(\mathcal{M}^*(n, n/L, n \mid L/\varepsilon)) \leq \tilde{O}((1/\varepsilon)Ln^\omega)$ time. For each (u, v) , we remember which z gives the minimum for the two products, and take the one with the smaller $\hat{D}[u, z] + \hat{D}[z, v]$ among the two.

To justify correctness, consider a pair u, v with $D[u, v] \geq L$. Consider a shortest path π from u to v . There exists a vertex $z^* \in R$ among the first L vertices in π . Thus, $D[u, z^*] \leq L$. Furthermore, among all $z \in R$ with $D[u, z] \leq L + O(\varepsilon)$, we have $D[z, v]$ lying in an interval I of length $2L + O(\varepsilon)$, since the graph is undirected. For either $a = 0$ or $a = 5L$, the shifted interval $I + a$ would be completely contained in $[L, 9L]$ modulo $10L$, and so the minimum of $\hat{D}[u, z] + \hat{D}[z, v]$ would be correctly computed, with additive error $O(\varepsilon) + O(\varepsilon) = O(\varepsilon)$.

Total time. The overall running time (ignoring $\text{poly}(1/\varepsilon)$ factors) is

$$\tilde{O}(Bn^\omega + tn^\omega/B^{\omega-2} + n^3/(tL) + Ln^\omega).$$

Choosing $t = B^{\omega-1}$ and $L = B$ (noting that indeed $B \leq t$) gives $\tilde{O}(Bn^\omega + n^3/B^\omega)$. Finally, choosing $B = n^{(3-\omega)/(\omega+1)}$ gives $\tilde{O}(n^{\omega+(3-\omega)/(\omega+1)}) = O(n^{(3+\omega^2)/(\omega+1)}) = O(n^{2.559})$.

Standard techniques for generating witnesses for matrix products can be applied to recover the approximate shortest paths [14, 23].

► **Theorem 3.** *Given an undirected graph with n vertices and real edge weights in $[0, 1]$, we can solve the approximate APSP problem with additive error $O(\varepsilon)$ in $\tilde{O}(n^{(3+\omega^2)/(\omega+1)}) = O(n^{2.559})$ time, for any constant $\varepsilon > 0$.*

6 Final Remarks

It remains open whether an $\tilde{O}(n^2)$ -time algorithm for undirected real-weighted graphs is possible if $\omega = 2$, even with a large constant additive error.


Under the current bounds on matrix multiplication, could our $O(n^{2.559})$ result be further improved? At the moment, we don't know how to use rectangular matrix multiplication to speed up our algorithm. And we don't know either how to use rectangular matrix multiplication to speed up Yuster's $O(n^{2.687})$ -time algorithm for directed graphs [22] (ideally, to match up with Zwick's unweighted exact APSP algorithm [23], which has running time $O(n^{2.529})$ via the latest bounds on rectangular matrix multiplication [17]).

References


- 1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.
- 2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.
- 3 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.
- 4 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (α, β) -spanners. *ACM Trans. Algorithms*, 7(1):5:1–5:26, 2010. doi:10.1145/1868237.1868242.
- 5 Karl Bringmann, Marvin Künnemann, and Karol Węgrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 943–954, 2019.
- 6 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.
- 7 Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Trans. Algorithms*, 8(4):34:1–34:17, 2012. doi:10.1145/2344422.2344424.
- 8 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.
- 9 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Algorithms, reductions and equivalences for small weight variants of all-pairs shortest paths. *CoRR*, abs/2102.06181, 2021. To appear in ICALP'21. arXiv:2102.06181.
- 10 Shiri Chechik. New additive spanners. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 498–512, 2013. doi:10.1137/1.9781611973105.36.

- 11 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.
- 12 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.
- 13 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. doi:10.1137/0205006.
- 14 Zvi Galil and Oded Margalit. Witnesses for Boolean matrix multiplication and for transitive closure. *J. Complex.*, 9(2):201–221, 1993.
- 15 Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *J. Comput. Syst. Sci.*, 54(2):243–254, 1997. doi:10.1006/jcss.1997.1385.
- 16 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Lukaszewicz, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs LCA in DAGs: Breaking through the $O(n^{2.5})$ barrier. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289, 2021. doi:10.1137/1.9781611976465.18.
- 17 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018.
- 18 Liam Roditty and Asaf Shapira. All-pairs shortest paths with a sublinear additive error. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP), Part I*, pages 622–633, 2008.
- 19 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.
- 20 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM)*, pages 3447–3487, 2018.
- 21 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.
- 22 Raphael Yuster. Approximate shortest paths in weighted graphs. *J. Comput. Syst. Sci.*, 78(2):632–637, 2012.
- 23 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

Dynamic Colored Orthogonal Range Searching

Timothy M. Chan 

Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Zhengcheng Huang 

Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract

In the *colored orthogonal range reporting* problem, we want a data structure for storing n colored points so that given a query axis-aligned rectangle, we can report the distinct colors among the points inside the rectangle. This natural problem has been studied in a series of papers, but most prior work focused on the static case. In this paper, we give a dynamic data structure in the 2D case which can answer queries in $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ time, where k denotes the output size (the number of distinct colors in the query range), and which can support insertions and deletions in $O(\log^{2+o(1)} n)$ time (amortized) in the standard RAM model. This is the first fully dynamic structure with polylogarithmic update time whose query cost per color reported is *sublogarithmic* (near $\sqrt{\log n}$). We also give an alternative data structure with $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$ query time and $O(\log^{3/2+o(1)} n)$ update time (amortized). We also mention extensions to higher constant dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases Range searching, dynamic data structures, word RAM

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.28

Funding Supported by NSF Grant CCF-1814026.

Acknowledgements We thank Saladi Rahul for helpful discussions.

1 Introduction

Range searching is one of the most fundamental data structure problems studied in computational geometry. Motivated by applications in databases and information retrieval, many researchers [5, 6, 7, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 29, 30, 31, 32, 34] have investigated a natural *colored* variant of range searching (also called “categorical”, or “generalized”, range searching):

Given a set of n points where each point is assigned a color (or “category”), we want to quickly report the distinct colors of the points inside a query range.

The query time should depend on the output size k , i.e., the number of distinct colors in the range (which could be much smaller than the number of points in the range). In this paper, we focus on the basic case of 2D *orthogonal* query ranges, i.e., axis-aligned rectangles.

A long series of work have studied this colored orthogonal range reporting problem, which turns out to be more challenging than the traditional uncolored problem; see Table 1 for a quick summary. Throughout the paper, we assume the standard word RAM model of computation.

All these prior papers addressed primarily static data structures, and surprisingly, not much progress has been made on the equally fundamental *dynamic* problem, where insertions and deletions of points are allowed.



© Timothy M. Chan and Zhengcheng Huang;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 28; pp. 28:1–28:13

Leibniz International Proceedings in Informatics



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

■ **Table 1** Static data structures for colored 2D orthogonal range reporting. (Coordinates are assumed to be integers bounded by U .)

	space	query time
Janardan & Lopez ('93) [21]	$O(n \log n)$	$O(\log^2 n + k)$
	$O(n \log^2 n)$	$O(\log n + k)$
Gupta, Janardan, & Smid ('95) [17]	$O(n \log^2 n)$	$O(\log n + k)$
Agarwal et al. (ESA'02) [1]	$O(n \log^2 n)$	$O(\log \log U + k)$
Mortensen ('03) [27]	$O(n \log n \log \log n)$	$O(\log^2 \log U + k)$
	$O(n \log n)$	$O(\log n \log^2 \log n + k)$
Shi & JaJa ('05) [34]	$O(n \log n)$	$O(\log n + k)$
Larsen & van Walderveen (SODA'13) [25]	$O(n \log n)$	$O(\log \log U + k)$
Nekrich (PODS'12) [30]	$O(n \log n)$	$O(\log \log U + k)$
Chan and Nekrich (SODA'20) [7]	$O(n \log^{3/4+\varepsilon} n)$	$O(\log \log U + k)$

Known dynamic colored results. The 1D dynamic colored range reporting problem – in which query ranges are intervals – can be easily reduced to 2D uncolored 3-sided range reporting (e.g., see the survey [16, Section 1.3.1]), and so by “textbook” *range trees* [11, 33], the problem can be solved with $O(\log n + k)$ query time and $O(\log^2 n)$ update time, or by incorporating dynamic fractional cascading [26], $O(\log n \log \log n + k)$ query time and $O(\log n \log \log n)$ update time. By using the current best results of Chan and Tsakalidis [9] on dynamic 2D uncolored range searching, the query time improves to $O(\frac{\log n}{\log \log n} + k)$ and the update time improves to $O(\log^{1/2+\varepsilon} n)$ (amortized) for an arbitrarily small constant $\varepsilon > 0$. (For dynamic 2D uncolored range searching, a lower bound of $\Omega(\frac{\log n}{\log \log n} + k)$ query time is known [2] for data structures with polylogarithmic update time; and $\sqrt{\log n}$ update time has also been recognized as a barrier that would be difficult to break with existing techniques [9].)

Straightforwardly, the dynamic 2D colored range reporting can be reduced dynamic 1D colored range reporting, by using a range tree on the x -coordinates, with a logarithmic-factor increase in both the query and update time. This implies a 2D colored data structure with $O(\frac{\log^2 n}{\log \log n} + k \log n)$ query time and $O(\log^{3/2+\varepsilon} n)$ update time. Notice the extra logarithmic factor in the $O(k \log n)$ term of the query cost: as the query range is decomposed into $O(\log n)$ sub-ranges, the same color may be discovered $O(\log n)$ times with this approach.

Alternatively, it is known (e.g., see [16, Section 3.4]) that a colored range reporting query can be reduced to $O(k \log n)$ number of uncolored range emptiness queries, by using a range tree on the colors. This simple reduction works in the dynamic setting, with update time increased by a logarithmic factor. By Chan and Tsakalidis’s result [9] on dynamic uncolored range emptiness, this implies a different dynamic 2D colored data structure with $O(k \frac{\log^2 n}{\log \log n})$ query time and $O(\log^{3/2+\varepsilon} n)$ update time, which is no better than the above.

Known static methods managed to avoid extra logarithmic factors in the k term (as can be seen in Table 1), but these methods do not adapt well to the dynamic setting. Some results were known in the insertion-only case [17], but it is open whether there exists a fully dynamic data structure with $O(\text{polylog } n + k)$ query time and $O(\text{polylog } n)$ update time for 2D colored orthogonal range reporting.¹

¹ However, it is not difficult to obtain $O(\frac{\log n}{\log \log n} + k)$ query time with $O(n^\varepsilon)$ update time, by modifying the reduction from 2D colored to 1D colored to use a larger fan-out $n^{\varepsilon'}$.

New dynamic colored result. We make progress towards this open problem by presenting a new data structure for dynamic 2D colored orthogonal range reporting with $O(\log n + k \log^{1/2+o(1)} n)$ query time and $O(\text{polylog } n)$ update time. The query cost per reported color is thus sublogarithmic ($\log^{1/2+o(1)} n$).

Interestingly, this improved query time bound is obtained using an approach similar to that in Chan and Pătraşcu’s $O(n\sqrt{\log n})$ -time inversion-counting algorithm [8], or in other known data structures with “fractional-power-of-log” update times [8, 9, 10, 28, 36]. As reinterpreted in Chan and Tsakalidis’ framework [9], the approach roughly involves two parts: (i) the design of *micro-structures* for small input (of size near $2^{\sqrt{\log n}}$), which uses bit packing tricks; and (ii) the design of *macro-structures*, which uses more traditional range trees but with larger fan-out (also near $2^{\sqrt{\log n}}$) to reduce the tree depth. What is novel in our application is that the $\sqrt{\log n}$ factor appears in the k term of the query cost – we will apply bit packing to the query’s output list of colors in the micro-structures. In addition, we follow an idea from Chan and Nekrich’s static data structure for colored range reporting [7]: first solve a “capped” version of the problem, where the number of reported colors k is promised to be upper-bounded by some parameter K_0 , and then extend the solution to general k by building a range tree on the colors. The overall method is conceptually not complicated, as explained in Section 2.

Improving the update time. In its simplest version, our data structure has $O(\log^{3+o(1)} n)$ update time. In Section 3, by combining with the more sophisticated techniques in Chan and Tsakalidis’ work on dynamic uncolored orthogonal range searching [9], we further reduce the update time to $O(\log^{2+o(1)} n)$, while keeping roughly the same query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$. These bounds are amortized.

Alternative result. In Section 4, by using a different micro-structure, we also describe an alternative data structure which further lowers the update time to $O(\log^{3/2+o(1)} n)$, although the query time is increased to $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$. Still, compared to the aforementioned prior result with $O(\frac{\log^2 n}{\log \log n} + k \log n)$ query time and $O(\log^{3/2+\varepsilon} n)$ update time, we get a strictly better query time bound while keeping the same update time.

Higher dimensions. Our approach can also be extended to higher dimensions. In Section 5, we show how to answer colored orthogonal range reporting queries in a constant dimension d in $O(\log^{d-1} n + k \log^{d-2+1/d+o(1)} n)$ time, while supporting updates in $O(\text{polylog } n)$ time. The $\log^{d-2+1/d} n$ factor is intriguingly similar to Chan and Pătraşcu’s bound on d -dimensional (uncolored) orthogonal range counting [8].

Computational model. We work with the standard w -bit word RAM model of computation, with $w = \Omega(\log n)$. On occasion, we assume that certain nonstandard word operations take unit time. This assumption may be removed by simulating such operations via table lookup, after an initial preprocessing of the table in $2^{O(w)}$ time, the cost of which is negligible since we will eventually set $w = \delta \log n$ for a sufficiently small constant $\delta > 0$.

2 First Method

In this section, we present our first method for 2D colored orthogonal range reporting, achieving roughly $O(\log n + k\sqrt{\log n})$ query time with polylogarithmic update time.

We begin by solving the case of *3-sided* query ranges, i.e., rectangles that are unbounded on one side, w.l.o.g., from below. The colored 3-sided problem is already challenging (and, in fact, so is 2-sided). The general idea is to build the data structure in two stages. First, in Section 2.1 we design a *micro-structure* to solve the problem for small input size, in which case the colors can be encoded in much fewer bits than $\log n$ and we can apply bit packing tricks. In Sections 2.2–2.3 we then use this micro-structure to build a *macro-structure*, solving the 3-sided problem for large input size. Finally, in Section 2.4, we note how the original 4-sided problem can be reduced to the 3-sided problem, without increasing the query time (though the update time is increased by an extra logarithmic factor).

2.1 3-Sided Micro-Structure

Our micro-structure is obtained by modifying the binary range tree and incorporating bit packing techniques.

► **Lemma 1.** *Given a set of at most s points in $2D$, there is a data structure for colored 3-sided range reporting with*

- *query time $Q_3(s, k) = O(\log s + (k \log k \log^{2+o(1)} s)/w + k)$, and*
- *amortized update time $U_3(s) = O(\log^{1+o(1)} s)$.*

Proof. Since there are at most s points and thus at most s colors, we can map each color to an integer in $\{1, \dots, s\}$. This enables us to pack $w/\log s$ colors into a single word. The mapping can be stored in a “color translation” table of s entries, so that we can translate each mapped color back to its original color in constant time. In an insertion of a point with a new color, we can simply map its color to the next unused integer.

The micro-structure takes the form of a binary range tree. At each node, we divide the plane into two vertical slabs, each with roughly half the number of points, such that each child node stores a recursive data structure for one of the slabs. For each slab σ , we store a list L_σ of all points in σ , sorted by the y -coordinates. In addition, we store a sublist L'_σ that contains the lowest point of each color in L_σ , also sorted by y . Using a colored predecessor search structure [28, Theorem 14], which has $O(\log^2 \log s)$ update time, for any given point, we can find the predecessor in L_σ and L'_σ in $O(\log^2 \log s)$ time, given its predecessor in L_τ of the parent slab τ ; we can also find the predecessor of a specific color in $O(\log^2 \log s)$ time.

When a point p is inserted to L_σ , if p has no predecessor of the same color, we insert it to L'_σ and delete its colored successor (if one exists) from L'_σ . When a point p is deleted from L_σ , if p is in L'_σ , we delete it and insert its colored successor to L'_σ .

By bit packing, we store the list of colors in L_σ in a sequence of words, each containing between $w/(2 \log s)$ and $w/\log s$ colors. During an insertion, when the number of elements in a word exceed $w/\log s$, we split the word into two. During a deletion, when the number of elements in a word drops below $w/(2 \log s)$, we merge the word with its successor in the sequence, and if the number of elements in the merged word exceeds $w/\log s$, we split it. This takes constant time (using standard word operations such as shifts).

We can ensure that the range tree has $O(\log s)$ height, for example, by known weight-balancing techniques [4]. The overall update time is thus $O(\log s \log^2 \log s)$.

For a given 3-sided query range q , if q intersects only one slab, then we recurse in that slab; otherwise, we make a recursive 2-sided query in each slab. Given a 2-sided query range q open to the left, if the vertical side of q intersects the left slab, then we recurse in the left slab; otherwise, we report all distinct colors in the left slab σ by scanning its sublist L'_σ from the bottom and extracting a prefix, and then recurse in the right slab. We can handle 2-sided query ranges open to the right symmetrically.

Each color is reported only once at each level, and therefore is reported $O(\log s)$ times over the recursion. Thus, the combined list of reported colors requires $O(k \log s \cdot \log s)$ bits and can be represented in $O(1 + (k \log^2 s)/w)$ words. We then remove duplicate colors by sorting and performing a linear scan. A bit-packed version of mergesort on m elements takes time $O(\log m)$ times the number of words (using possibly nonstandard word operations). Thus, the cost is $O(\log s + (1 + (k \log^2 s)/w) \cdot \log(k \log s)) = O(\log s + (k \log k \log^{2+o(1)} s)/w)$. Afterwards, we spend $O(k)$ time to translate the output colors. ◀

2.2 3-Sided Capped Macro-Structure

Next, we present our macro-structure for large input. We first solve the K_0 -capped problem. Here, the number of colors in the query range is promised to be smaller than a given fixed value K_0 , i.e., $k \leq K_0$.

► **Lemma 2.** *Fix a value $K_0 \leq 2^{\sqrt{w}}$. Given n points in $2D$, there is a data structure for colored 3-sided K_0 -capped range reporting with*

- *query time $O(\log n + (k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k) \leq O(\log n + k \log K_0 \log^{1/2+o(1)} n)$;*
- *amortized update time $O(\log^{1+o(1)} n)$.*

Proof. We will still use a variant of the range tree, but with a much larger fan-out s to be set later. At each node, the plane is divided into s vertical slabs, each with about $1/s$ of the points. For each slab σ , among the lowest point of each color, we only take the K_0 lowest ones. We replace the x -coordinates of these points with that of the left side of σ . We then store all sK_0 such points across all slabs in the micro-structure from Lemma 1, with $U_3(sK_0)$ update time and $Q_3(sK_0, k)$ query time. We maintain colored predecessor search structures at each node as before. We can ensure that the range tree has $O(\log_s n)$ height, by known weight-balancing techniques [4].

For a given 3-sided query range q , we identify the child slabs σ_1 and σ_2 containing the two vertices of q , answer a query for the micro-structure for $q - \sigma_1 - \sigma_2$, and then recurse in σ_1 and σ_2 . We make $O(\log_s n)$ queries on the micro-structures along two paths of the tree, so the query time is

$$Q'_3(n, k) = O(Q_3(sK_0, k) \log_s n) = O\left(\log(sK_0) + k \frac{\log k \log^{2+o(1)}(sK_0)}{w} + k\right) \cdot \log_s n.$$

For each update, we make $O(\log_s n)$ updates on the micro-structures along a path of the tree, so the update time is

$$U'_3(n) = O(U_3(sK_0) \log_s n + \log_s n \log^2 \log n) = O((\log^{1+o(1)}(sK_0) + \log^2 \log n) \log_s n).$$

Setting $s = 2^{\sqrt{w}}$ and recalling that $K_0 \leq 2^{\sqrt{w}}$, we obtain $Q'_3(n, k) = O(\log n + (k \log k \log^{1+o(1)} n)/\sqrt{w} + k)$ and $U'_3(n) = O(\log^{1+o(1)} n)$. ◀

Remark. If $k \geq K_0$, the algorithm may or may not succeed in reporting all k distinct colors. It is actually possible to *pre-check* whether the algorithm will succeed, in $O(\log n)$ time (without paying the $O((k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k)$ cost): In the data structure from Lemma 2, recall that in each slab we store the lowest K_0 points of distinct colors in a micro-structure. We mark the K_0 -th lowest point with a special color. In the micro-structure from Lemma 1, in each L_σ , we maintain the lowest special point. If the query algorithm wants to report a prefix of L'_σ , we check that the lowest special point is not in the query range (otherwise the algorithm would fail). We also check that the K_0 -th lowest point of L'_σ is not in the query range. The cost of pre-checking is $O(\log_s n \cdot \log(sK_0)) = O(\log n)$.

2.3 From Capped to Uncapped

We now remove the assumption $k < K_0$, to obtain a complete solution to the 3-sided problem:

► **Theorem 3.** *Given n points in $2D$, there is a data structure for colored 3-sided range reporting with*

- *query time $O(\log n + (k \log^{1+o(1)} n)/\sqrt{w} + k) \leq O(\log n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{2+o(1)} n)$.*

Proof. We use a range tree on the colors, as in Chan and Nekrich’s static colored method [7, proof of Theorem 2.3], with fan-out f . At each node we store its point set in the capped data structure from Lemma 2. The color class is partitioned into f color subclasses, each with roughly equal number of colors. Each child of a node corresponds to one of these subclasses. We can ensure that the range tree has $O(\log_f n)$ height, by known weight-balancing techniques [4].

To answer a query, at each node we first pre-check whether the query will succeed, as in the Remark after Lemma 2.² If so, we query the capped structure from Lemma 2 at the node and finish the query. Otherwise, we recurse in all f children.

Whenever we recurse in the children of a node, the node must contain $\geq K_0$ colors in the query range. Thus, the number of recursive calls per level is $O(fk/K_0)$, so the total number of recursive calls is $O((fk/K_0) \log_f n)$, excluding the root. Thus, the total cost of pre-checking is $O((fk/K_0) \log_f n \cdot \log n)$. The total query time spent on capped structures is $O(((fk/K_0) \log_f n + 1) \cdot \log n + (k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k)$.

For each update, we make $O(\log_f n)$ updates to the capped structures along a path of the tree, so the total update time is $O(\log_f n \cdot \log^{1+o(1)} n)$.

Setting $f = 2$ and $K_0 = \log^2 n$ yields query time $O(\log n + (k \log^{1+o(1)} n)/\sqrt{w} + k)$ and update time $O(\log^{2+o(1)} n)$. ◀

2.4 From 3-Sided to 4-Sided

By building a standard binary range tree where each node stores the 3-sided structure from Theorem 3, we can reduce a 4-sided query to two 3-sided queries. The query time remains the same, and the update time increases by a logarithmic factor. This immediately yields the following result:

► **Corollary 4.** *Given n points in $2D$, there is a data structure for colored 4-sided range reporting with*

- *query time $O(\log n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{3+o(1)} n)$.*

3 Improving the 4-Sided Update Time

In this section, we apply more advanced techniques to improve the update time for 4-sided queries to $O(\log^{2+o(1)} n)$, which matches our 3-sided update time, avoiding the logarithmic-factor penalty in going from 3-sided to 4-sided.

Chan and Tsakalidis [9] proposed a framework for transforming micro-structures into macro-structures for uncolored range searching. We observe that their transformation works for the capped colored problem as well, with minor modifications. Our improved colored

² Chan and Nekrich [7] originally suggested some nontrivial approximate range counting approach to do the pre-checking, which we have bypassed.

result then follows by combining this transformation with our 3-sided micro-structure from Section 2.1, and with the known transformation of capped to uncapped structures used in Section 2.3.

In more details, Chan and Tsakalidis started with a technique of Mortensen [28, Theorem 1], which they dubbed the *van Emde Boas transformation*, for converting a micro-structure to a data structure for solving the *narrow grid* case, where input points have a small number of distinct x -coordinates. The conversion increases time bounds by only $\log \log$ factors, and is achieved by a recursion similar to van Emde Boas trees [35]. We observe that a similar transformation holds for the capped colored range reporting problem, if the cap K_0 is not too big. The statement below is adapted from Chan and Tsakalidis [9, Lemma 4]. In this section, 3-sided query ranges are unbounded from the left or the right (instead of from below).

► **Lemma 5.** *Fix K_0 . Let X be a set of $O(s)$ values. Given a dynamic data structure for colored j -sided range reporting ($j \in \{3, 4\}$) on s' points in $X \times \mathbb{R}$ with query time $Q_j(s, s', k)$ and update time $U_j(s, s')$ (amortized), there is a data structure for colored j -sided K_0 -capped range reporting on n points in $X \times \mathbb{R}$ with (amortized)*

- query time $Q_j(s, n, k) = O(Q_j(s, (sK_0)^{O(1)}, k) \log \log n)$, and
- update time $U_j(s, n) = O(U_j(s, (sK_0)^{O(1)}) \log^2 \log n)$.

If the given data structure supports updates to X in $U_X(s)$ time and this update procedure depends solely on X (and not the point set), the new data structure can support updates to X in $U_X(s)$ time.

The proof is a straightforward modification of the previous proof [28, 9]. (Basically, whenever in the uncolored solution we refer to the topmost/bottommost point, we now consider the K_0 topmost/bottommost points of distinct colors – this explains why the number of points in the micro-structures goes up from $s^{O(1)}$ to $(sK_0)^{O(1)}$.)

Next, Chan and Tsakalidis [9, Lemma 7] described a way to convert a narrow-grid structure into a data structure for the general case. This transformation, which they dubbed the (first) *range tree transformation*, uses standard range trees with fan-out s . We observe that the same transformation works for the capped colored problem:

► **Lemma 6.** *Fix K_0 . Given a family of data structures $\mathcal{D}_j^{(i)}$ ($i \in \{1, \dots, \log_s n\}$) for dynamic colored j -sided K_0 -capped range reporting ($j \in \{3, 4\}$) on n points in $X \times \mathbb{R}$ ($|X| = O(s)$) with query time $Q_j^{(i)}(s, n, k)$ update time $U_j^{(i)}(s, n)$ (amortized), where updates to X take $U_X(s)$ time, there is a data structure for colored 4-sided K_0 -capped range reporting on n points in the plane with the following query and update time (amortized):*

$$Q'_4(n, k) = O\left(\max_i Q_4^{(i)}(s, n, k) + \max_i Q_3^{(i)}(s, n, k) \log_s n + \log_s n \log^2 \log n\right)$$

$$U'_4(n) = O\left(\sum_{i=1}^{\log_s n} (U_4^{(i)}(s, n) + U_3^{(i)}(s, n)) + \sum_{i=1}^{\log_s n} \frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n \log^2 \log n\right).$$

The proof of the above requires essentially no change in the previous proof [9].

We are now ready to put together our new improved data structure. We set $K_0 = s$. Our colored 3-sided method in Theorem 3 gives

$$Q_3(s, s^{O(1)}, k) = O(\log s + (k \log^{1+o(1)} s) / \sqrt{w} + k)$$

$$U_3(s, s^{O(1)}) = O(\log^{2+o(1)} s).$$

The known colored 4-sided method mentioned in the introduction gives

$$\begin{aligned} Q_4(s, s^{O(1)}, k) &= O\left(\frac{\log^2 s}{\log \log s} + k \log s\right) \\ U_4(s, s^{O(1)}) &= O(\log^{3/2+\varepsilon} s). \end{aligned}$$

(Actually, a weaker bound of $U_4(s, s^{O(1)}) = O(\log^{2+o(1)} s)$ suffices.) In both of these methods, $U_X(s) = 0$. Applying Lemmas 5–6 (with $Q_4^{(i)} = Q_4$ and $Q_3^{(i)} = Q_3$ for all i) yields

$$\begin{aligned} Q'_4(n, k) &= O\left(\frac{\log^2 s}{\log \log s} + k \log s + (\log s + (k \log^{1+o(1)} s)/\sqrt{w} + k) \log_s n\right) \cdot \log^2 \log n \\ U'_4(n) &= O(\log^{2+o(1)} s \log_s n) \cdot \log^2 \log n. \end{aligned}$$

Setting $s = 2^{\sqrt{\log n}}$ gives

$$\begin{aligned} Q'_4(n, k) &= O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n) \\ U'_4(n) &= O(\log^{3/2+o(1)} n). \end{aligned}$$

All this is for the K_0 -capped problem with $K_0 = s = 2^{\sqrt{\log n}}$. Finally, we solve the general uncapped problem by the color range tree technique in Section 2.3. As in the Remark after Lemma 2, it is possible to pre-check whether the query will succeed, in $O(\log^{1+o(1)} n)$ time (without paying the $O(k \log^{1/2+o(1)} n)$ cost); the modification is straightforward (see the Appendix regarding the van Emde Boas transformation). As in the proof of Theorem 3, by using a range tree on colors with fan-out f , the overall query time is $O(((fk/K_0) \log_f n + 1) \cdot \log^{1+o(1)} n + k \log^{1/2+o(1)} n)$, and the overall update time is $O(\log_f n \log^{3/2+o(1)} n)$.

By setting $f = \sqrt{K_0}$ for example, the query time remains $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ and the update time becomes $O(\log^{2+o(1)} n)$.

► **Theorem 7.** *Given n points in 2D, there is a data structure for colored 4-sided range reporting with*

- *query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{2+o(1)} n)$.*

4 Alternative Method

We now describe an alternative method that has smaller update time $O(\log^{3/2+o(1)} n)$, though the query time is increased to $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$.

The solution uses a different micro-structure, given in the lemma below, which works directly for the general 4-sided case. Its main advantage is that it has *constant* amortized update time when the input size s is sufficiently small (around $2^{\log^{1/4} n}$). It is a colored variant of a micro-structure for uncolored 4-sided queries by Chan and Tsakalidis [9, Lemma 5(ii) and Lemma 6].

► **Lemma 8.** *Fix a parameter $\bar{w} \leq w$ with $\bar{w} = \Omega(\log s)$. Given a set of at most s points in a universe $X \times Y$ with $|X|, |Y| = s^{O(1)}$, and there is a data structure for 4-sided colored range reporting with*

- *amortized query time $O(\log^2 s + (k \log^{3+o(1)} s)/\bar{w} + k)$, and*
- *amortized update time $O(1 + (\log^4 s)/\bar{w})$.*

In addition, it supports:

- *updates to Y in $O(\log^2 \log n)$ time, and*
- *updates to X in $2^{O(\bar{w})}$ time, where the update procedure depends solely on X (and not the point set).*

Proof. Recall that colors can be mapped to $(\log s)$ -bit integers, as noted in the first paragraph of the proof of Lemma 1.

We follow the approach by Chan and Tsakalidis [9, Lemma 5(ii)] and mimic an *external-memory* data structure with block size $B := \delta \bar{w} / \log s$ for a sufficiently small constant δ . A block of B points can be encoded in $O(\delta \bar{w})$ bits and can thus be packed in a single word.

The well-known *buffer tree* of Arge [3] is a 1D external-memory data structure with subconstant ($O(\frac{1}{B} \log_B s)$) amortized update cost. For their 4-sided uncolored micro-structure, Chan and Tsakalidis suggested an analogous *buffered* version of the binary range tree in 2D, which has amortized update cost $O(\frac{1}{B} \log^2 s) = O((\log^3 s)/\bar{w})$. Our solution in the colored setting is similar.

Roughly speaking, in the primary 2D range tree, each node corresponds to a *canonical horizontal slab* and stores a secondary tree, which is a 1D range tree of the points in the corresponding canonical horizontal slab. In a secondary tree, each node corresponds a *canonical rectangle*. In the buffered version, each node of the primary tree and secondary trees holds a buffer of up to B update requests that are yet to be processed.

We will not re-explain all the details of the buffered 2D range tree, but just describe the main changes needed for the colored problem: For each node ν of a secondary tree, we additionally maintain a sorted list L_ν of the distinct colors appearing in the corresponding canonical rectangle, along with the multiplicity of these colors. We also hold a buffer Z_ν of (a possibly large number of) of update requests yet to be processed at ν . Updates in ν are handled lazily, by just appending the requests to the buffer Z_ν . We postpone the work of actually updating L_ν to querying. When a query wants to report the colors in a node ν , we sort Z_ν by color and perform a linear scan to update L_ν . Let ℓ_ν and z_ν be the number of elements in L_ν and Z_ν . The lists L_ν and Z_ν can be packed in $O((\ell_\nu \log s)/w)$ and $O(z_\nu \log s/\bar{w})$ words respectively. By a bit-packed version of mergesort, sorting Z_ν takes time $O((z_\nu \log s)/\bar{w}) \cdot \log s$. Since an update leads to update requests in the buffers Z_ν of $O(\log^2 s)$ nodes ν , we can assign an amortized cost of $O((\log^4 s)/\bar{w})$ per update to cover this sorting cost. Scanning L_ν takes time $O((\ell_\nu \log s)/w)$. Now, ℓ_ν is bounded by the number of colors reported in ν during the previous query that involves ν . Since a query visits $O(\log^2 s)$ nodes, we can assign an amortized cost of $O((k \log^3 s)/w)$ per query with output size k to cover this scanning cost.

In a query, we report the colors in the sorted list L_ν for $O(\log^2 s)$ nodes ν . The combined list of reported colors has $O(k \log^2 s)$ colors and requires $O((k \log^3 s)/\bar{w})$ words. We remove duplicate colors by merging these $O(\log^2 s)$ sorted lists. This multi-way merging can be done in $O(\log \log s)$ rounds, each taking time linear in the number of words. Thus, the cost is $O((k \log^{3+o(1)} s)/\bar{w})$. Afterwards, we spend $O(k)$ time to translate the output colors.

Updates to X and Y can be handled using the same ideas by Chan and Tsakalidis [9, Lemma 6]. ◀

We can now put together the new data structure by combining the new micro-structure with Chan and Tsakalidis' framework [9], via the van Emde Boas transformation and range tree transformation, as we have described in Section 3.

As before, we set $K_0 = s$. For each $j \in \{3, 4\}$ and each $i \in \{1, \dots, \log_s n\}$, we apply Lemma 8 with the choice $\bar{w} = \delta i \log s$ to get

$$\begin{aligned} Q_j^{(i)}(s, s^{O(1)}, k) &= O(\log^2 s + (k \log^{2+o(1)} s)/i + k) \\ U_j^{(i)}(s, s^{O(1)}) &= O(1 + (\log^{3+o(1)} s)/i) \\ U_X^{(i)}(s) &= s^{O(\delta i)}. \end{aligned}$$

28:10 Dynamic Colored Orthogonal Range Searching

Applying Lemmas 5–6 and noting that $\sum_{i=1}^{\log_s n} 1/i = O(\log \log_s n)$ yields

$$\begin{aligned} Q'_4(n, k) &= O\left(\log^2 s \log_s n + k \log^{2+o(1)} s + k \log_s n\right) \cdot \log^3 \log n \\ U'_4(n) &= O(\log_s n + \log^{3+o(1)} s) \cdot \log^3 \log n. \end{aligned}$$

Setting $s = 2^{\log^{1/4} n}$ gives

$$\begin{aligned} Q'_4(n, k) &= O(\log^{5/4+o(1)} n + k \log^{3/4+o(1)} n) \\ U'_4(n) &= O(\log^{3/4+o(1)} n). \end{aligned}$$

All this is for the K_0 -capped problem with $K_0 = s = 2^{\log^{1/4} n}$. Finally, we solve the general uncapped problem by using the color range tree technique in Section 2.3. As before, it is possible to pre-check whether the query will succeed, in $O(\log^{5/4+o(1)} n)$ time (without paying the $O(k \log^{3/4+o(1)} n)$ cost). The overall query time is then $O(((fk/K_0) \log_f n + 1) \cdot \log^{5/4+o(1)} n + k \log^{3/4+o(1)} n)$, and the overall update time is $O(\log_f n \log^{3/4+o(1)} n)$.

By setting $f = \sqrt{K_0}$ for example, the query time remains $O(\log^{5/4+o(1)} n + k \log^{3/4+o(1)} n)$ and the update time becomes $O(\log^{3/2+o(1)} n)$.

The term $O(\log^{5/4+o(1)} n)$ dominates the query time bound only when $k \ll \sqrt{\log n}$, but in this case, we can switch to the method in Section 3, which solves the problem for an even bigger cap $2\sqrt{\log n}$ with a better query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ while keeping update time $O(\log^{3/2+o(1)} n)$.

► **Theorem 9.** *Given n points in 2D, there is a data structure for colored 4-sided range reporting with*

- *amortized query time $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$, and*
- *amortized update time $O(\log^{3/2+o(1)} n)$.*

5 Higher Dimensions

We now point out a generalization of the method in Section 2 to higher dimensions.

First, it is straightforward to generalize the micro-structure from Lemma 1 (here, a $(2d-1)$ -sided range is unbounded along the d -th axis):

► **Lemma 10.** *Given a set of at most s points in a constant dimension d , there is a data structure for colored $(2d-1)$ -sided range reporting with*

- *query time $Q_{2d-1}(s, k) = O(\log^{d-1} s + (k \log k \log^{d+o(1)} s)/w + k)$, and*
- *amortized update time $U_{2d-1}(s) = O(\log^{d-1+o(1)} s)$.*

We generalize the capped macro-structure from Lemma 2:

► **Lemma 11.** *Fix a value $K_0 \leq 2^{w^{1/d}}$. Given n points in a constant dimension d , there is a data structure for colored $(2d-1)$ -sided K_0 -capped range reporting with*

- *query time $O(\log^{d-1} n + k \log K_0 \log^{d-2+1/d+o(1)} n)$;*
- *amortized update time $O(\log^{d-1+o(1)} n)$.*

Proof. For each $j \in \{0, \dots, d-1\}$, define \mathcal{P}_j to be the (capped) subproblem in d dimensions which the first j coordinates of all points come from a set X of $O(s)$ values.

Following the proof of Lemma 2 of using a range tree but along the j -th axis, we can reduce problem \mathcal{P}_j to problem \mathcal{P}_{j+1} at the expense of increasing the query and update time by one $O(\log_s n)$ factor.

We solve problem \mathcal{P}_{d-1} directly: Points lie on $O(s^{d-1})$ vertical lines. For each such vertical lines, among the lowest point of each color, we only take the K_0 lowest points, and store all these $O(s^{d-1}K_0)$ points in the micro-structure from Lemma 10.

Thus, the original problem \mathcal{P}_0 can be solved with query and update time

$$\begin{aligned} Q'_{2d-1}(n, k) &= O(Q_{2d-1}(s^{d-1}K_0, k) \cdot (\log_s n)^{d-1}) \\ &= O\left(\log^{d-1}(sK_0) + k \frac{\log k \log^{d+o(1)}(sK_0)}{w} + k\right) \cdot (\log_s n)^{d-1} \\ U'_{2d-1}(n, K) &= O(U_{2d-1}(s^{d-1}K_0, k) \cdot (\log_s n)^{d-1}) = O(\log^{d-1+o(1)}(sK_0)) \cdot (\log_s n)^{d-1}. \end{aligned}$$

Setting $s = 2^{w^{1/d}}$ and recalling that $K_0 \leq 2^{w^{1/d}}$, we obtain $Q'_{2d-1}(n, k) = O(\log^{d-1} n + (k \log k \log^{d-1+o(1)} n)/w^{1-1/d} + k)$ and $U'_{2d-1}(n) = O(\log^{d-1+o(1)} n)$. ◀

We can solve the general uncapped problem by using the color range tree technique in Section 2.3 (invoking the capped structure with K_0 polylogarithmic). The update time is increased by a logarithmic factor:

► **Theorem 12.** *Given n points in a constant dimension d , there is a data structure for colored $(2d-1)$ -sided range reporting with*

- *query time $O(\log^{d-1} n + k \log^{d-2+1/d+o(1)} n)$, and*
- *amortized update time $O(\log^{d+o(1)} n)$.*

Finally, as in Section 2.4, general $(2d)$ -sided queries can be reduced to $(2d-1)$ -sided queries, with another logarithmic-factor increase in the update time:

► **Corollary 13.** *Given n points in a constant dimension d , there is a data structure for colored $(2d-1)$ -sided range reporting with*

- *query time $O(\log^{d-1} n + k \log^{d-2+1/d+o(1)} n)$, and*
- *amortized update time $O(\log^{d+1+o(1)} n)$.*

6 Final Remarks

All the extra $\log^{o(1)} n$ factors are $(\log \log n)^{O(1)}$. In the 2D query time bound of Corollary 4, the $O(\log n)$ first term is likely improvable slightly to $O(\frac{\log n}{\log \log n})$ (to match known lower bounds), by increasing the fan-out from 2 to $\log^\delta n$ in the proof of Lemma 1.

The main open question is whether $O(\text{polylog } n + k)$ query time is achievable with $O(\text{polylog } n)$ update time in 2D.

In higher dimensions, we have not tried to optimize the number of logarithmic factors in the update time, but the ideas in Section 3 should help. A more intriguing question is whether $O(\text{polylog } n + k \log^{d-2-\delta} n)$ query time is achievable with $O(\text{polylog } n)$ update time. Even for the static problem, we do not know how to get $O(\text{polylog } n + k)$ query time with $O(n \text{ polylog } n)$ space; current techniques only give $O(\text{polylog } n + k \log^{d-3} n)$ query time [7].

References

- 1 Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th Annual European Symposium on Algorithms (ESA)*, pages 17–28, 2002. doi:10.1007/3-540-45749-6_6.
- 2 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–544, 1998. doi:10.1109/SFCS.1998.743504.

- 3 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.
- 4 Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM J. Comput.*, 32(6):1488–1508, 2003.
- 5 Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–474, 1995. doi:10.1007/3-540-60084-1_97.
- 6 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further results on colored range searching. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:15, 2020. doi:10.4230/LIPIcs.SoCG.2020.28.
- 7 Timothy M. Chan and Yakov Nekrich. Better data structures for colored orthogonal range reporting. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–636, 2020. doi:10.1137/1.9781611975994.38.
- 8 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.
- 9 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:13, 2017. doi:10.4230/LIPIcs.SoCG.2017.28.
- 10 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic planar orthogonal point location in sublogarithmic time. In *Proc. 34th International Symposium on Computational Geometry (SoCG)*, pages 25:1–25:15, 2018. doi:10.4230/LIPIcs.SoCG.2018.25.
- 11 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
- 12 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct color searching in one dimension. In *Proc. 28th International Symposium on Algorithms and Computation (ISAAC)*, pages 30:1–30:11, 2017. doi:10.4230/LIPIcs.ISAAC.2017.30.
- 13 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J. Puglisi. Colored range queries and document retrieval. *Theor. Comput. Sci.*, 483:36–50, 2013. doi:10.1016/j.tcs.2012.08.004.
- 14 Arnab Ganguly, J. Ian Munro, Yakov Nekrich, Rahul Shah, and Sharma V. Thankachan. Categorical range reporting with frequencies. In *Proc. 22nd International Conference on Database Theory (ICDT)*, pages 9:1–9:19, 2019. doi:10.4230/LIPIcs.ICDT.2019.9.
- 15 Roberto Grossi and Søren Vind. Colored range searching in linear space. In *Proc. 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 229–240, 2014. doi:10.1007/978-3-319-08404-6_20.
- 16 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, chapter 67, pages 1042–1057. CRC Press, 2nd edition, 2018. URL: <https://www-users.cs.umn.edu/~sala0198/Papers/ds2-handbook.pdf>.
- 17 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *J. Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.
- 18 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Algorithms for generalized halfspace range searching and other intersection searching problems. *Comput. Geom.*, 6:1–19, 1996. doi:10.1016/0925-7721(95)00012-7.
- 19 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. A technique for adding range restrictions to generalized searching problems. *Inf. Process. Lett.*, 64(5):263–269, 1997. doi:10.1016/S0020-0190(97)00183-X.

- 20 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Algorithms for some intersection searching problems involving circular objects. *International Journal of Mathematical Algorithms*, 1:35–52, 1999.
- 21 Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.
- 22 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008. doi:10.1137/070684483.
- 23 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proc. 22nd ACM Symposium on Computational Geometry (SoCG)*, pages 52–60, 2006. doi:10.1145/1137856.1137866.
- 24 Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 583–592, 2012. URL: <http://portal.acm.org/citation.cfm?id=2095165-&CFID=63838676-&CFTOKEN=79617016>.
- 25 Kasper Green Larsen and Freek van Walderveen. Near-optimal range reporting structures for categorical data. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 256–276, 2013.
- 26 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990. doi:10.1007/BF01840386.
- 27 Christian Worm Mortensen. Generalized static orthogonal range searching in less space. Technical report, IT University Technical Report Series 2003-33, 2003.
- 28 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.
- 29 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002. doi:10.1145/545381.545469.
- 30 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9, 2014. doi:10.1145/2543924.
- 31 Yakov Nekrich and Jeffrey Scott Vitter. Optimal color range reporting in one dimension. In *Proc. 21st Annual European Symposium Algorithms (ESA)*, pages 743–754, 2013. doi:10.1007/978-3-642-40450-4_63.
- 32 Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proc. 33rd ACM Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014. doi:10.1145/2594538.2594557.
- 33 F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
- 34 Qingmin Shi and Joseph JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Inf. Process. Lett.*, 95(3):382–388, 2005. doi:10.1016/j.ipl.2005.04.008.
- 35 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.
- 36 Bryan T. Wilkinson. Amortized bounds for dynamic orthogonal range reporting. In *Proc. 22th Annual European Symposium on Algorithms (ESA)*, pages 842–856, 2014. doi:10.1007/978-3-662-44777-2_69.

ℓ_p -Norm Multiway Cut

Karthekeyan Chandrasekaran   

University of Illinois at Urbana-Champaign, IL, USA

Weihang Wang 

University of Illinois at Urbana-Champaign, IL, USA

Abstract

We introduce and study ℓ_p -NORM-MULTIWAY-CUT: the input here is an undirected graph with non-negative edge weights along with k terminals and the goal is to find a partition of the vertex set into k parts each containing exactly one terminal so as to minimize the ℓ_p -norm of the cut values of the parts. This is a unified generalization of min-sum multiway cut (when $p = 1$) and min-max multiway cut (when $p = \infty$), both of which are well-studied classic problems in the graph partitioning literature. We show that ℓ_p -NORM-MULTIWAY-CUT is NP-hard for constant number of terminals and is NP-hard in planar graphs. On the algorithmic side, we design an $O(\log^2 n)$ -approximation for all $p \geq 1$. We also show an integrality gap of $\Omega(k^{1-1/p})$ for a natural convex program and an $O(k^{1-1/p-\epsilon})$ -inapproximability for any constant $\epsilon > 0$ assuming the small set expansion hypothesis.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases multiway cut, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.29

Related Version Full Version: <https://arxiv.org/abs/2106.14840>

Funding Karthekeyan Chandrasekaran: Supported in part by NSF grants CCF 1814613 and 1907937. Weihang Wang: Supported in part by NSF grants CCF 1814613 and 1907937.

1 Introduction

MULTIWAY-CUT is a fundamental problem in combinatorial optimization with both theoretical as well as practical motivations. The input here is an undirected graph $G = (V, E)$ with non-negative edge weights $w : E \rightarrow \mathbb{R}_+$ along with k specified terminals $T = \{t_1, t_2, \dots, t_k\} \subseteq V$. The goal is to find a partition $\mathcal{P} = (P_1, P_2, \dots, P_k)$ of the vertex set with $t_i \in P_i$ for each $i \in [k]$ so as to minimize the sum of the cut values of the parts, i.e., the objective is to minimize $\sum_{i=1}^k w(\delta(P_i))$, where $\delta(P_i)$ denotes the set of edges with exactly one end-vertex in P_i and $w(\delta(P_i)) := \sum_{e \in \delta(P_i)} w(e)$. On the practical side, MULTIWAY-CUT has been used to model file-storage in networks as well as partitioning circuit elements among chips – see [14, 22]. On the theoretical side, MULTIWAY-CUT generalizes the min (s, t) -cut problem which is polynomial-time solvable. In contrast to min (s, t) -cut, MULTIWAY-CUT is NP-hard for $k \geq 3$ terminals [14]. The algorithmic study of MULTIWAY-CUT has led to groundbreaking rounding techniques and integrality gap constructions in the field of approximation algorithms [2, 4–7, 12, 16, 17, 21] and novel graph structural techniques in the field of fixed-parameter algorithms [18]. It is known that MULTIWAY-CUT does not admit a $(1.20016 - \epsilon)$ -approximation for any constant $\epsilon > 0$ assuming the Unique Games Conjecture [4] and the currently best known approximation factor is 1.2965 [21].

Motivated by its connections to partitioning and clustering, Svitkina and Tardos [22] introduced a local part-wise min-max objective for MULTIWAY-CUT – we will denote this problem as MIN-MAX-MULTIWAY-CUT: The input here is the same as MULTIWAY-CUT while the goal is to find a partition $\mathcal{P} = (P_1, P_2, \dots, P_k)$ of the vertex set with $t_i \in P_i$ for each $i \in [k]$ so as to minimize $\max_{i=1}^k w(\delta(P_i))$. We note that MULTIWAY-CUT and MIN-MAX-MULTIWAY-CUT differ only in the objective function – the objective function in MULTIWAY-CUT is to minimize



© Karthekeyan Chandrasekaran and Weihang Wang;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 29; pp. 29:1–29:15

Leibniz International Proceedings in Informatics



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

the sum of the cut values of the parts while the objective function in MIN-MAX-MULTIWAY-CUT is to minimize the max of the cut values of the parts. MIN-MAX-MULTIWAY-CUT can be viewed as a fairness inducing multiway cut as it aims to ensure that no part pays too much in cut value. Svitkina and Tardos showed that MIN-MAX-MULTIWAY-CUT is NP-hard for $k \geq 4$ terminals and also that it admits an $O(\log^3 n)$ -approximation. Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, and Schwartz subsequently improved the approximation factor to $O(\sqrt{\log n \log k})$ (which is $O(\log n)$) [3].

In this work, we study a unified generalization of MULTIWAY-CUT and MIN-MAX-MULTIWAY-CUT that we term as ℓ_p -NORM-MULTIWAY-CUT: In this problem, the input is the same as MULTIWAY-CUT, i.e., we are given an undirected graph $G = (V, E)$ with non-negative edge weights $w : E \rightarrow \mathbb{R}_+$ along with k specified terminal vertices $T = \{t_1, t_2, \dots, t_k\} \subseteq V$. The goal is to find a partition $\mathcal{P} = (P_1, P_2, \dots, P_k)$ of the vertex set with $t_i \in P_i$ for each $i \in [k]$ so as to minimize the ℓ_p -norm of the cut values of the k parts – formally, we would like to minimize

$$\left(\sum_{i=1}^k \left(\sum_{e \in \delta(P_i)} w(e) \right)^p \right)^{\frac{1}{p}}.$$

Throughout, we will consider $p \geq 1$. We note that ℓ_p -NORM-MULTIWAY-CUT for $p = 1$ corresponds to MULTIWAY-CUT and for $p = \infty$ corresponds to MIN-MAX-MULTIWAY-CUT. We emphasize that ℓ_p -NORM-MULTIWAY-CUT could also be viewed as a multiway cut that aims for a stronger notion of fairness than MULTIWAY-CUT but a weaker notion of fairness than MIN-MAX-MULTIWAY-CUT. For $k = 2$ terminals, ℓ_p -NORM-MULTIWAY-CUT reduces to min (s, t) -cut for all $p \geq 1$ and hence, can be solved in polynomial time.

1.1 Our Results

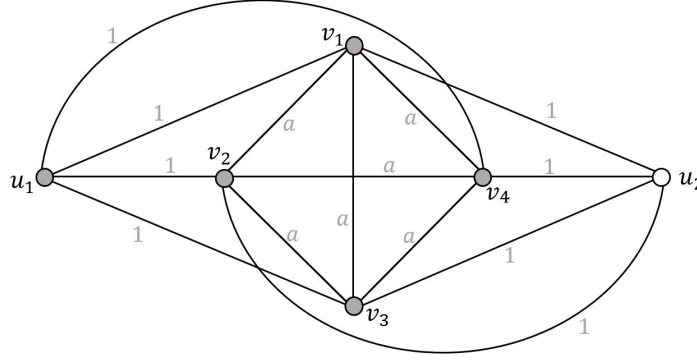
We begin by remarking that there is a fundamental structural difference between MULTIWAY-CUT and ℓ_p -NORM-MULTIWAY-CUT for $p > 1$ (i.e., between $p = 1$ and $p > 1$). The optimal partition to MULTIWAY-CUT satisfies a nice structural property: assuming that the input graph is connected, every part in an optimal partition for MULTIWAY-CUT will induce a connected subgraph. Consequently, MULTIWAY-CUT is also phrased as the problem of deleting a least weight subset of edges so that the resulting graph contains k connected components with exactly one terminal in each component. However, this nice structural property does not hold for ℓ_p -NORM-MULTIWAY-CUT for $p > 1$ as illustrated by the example in Figure 1. We remark that Svitkina and Tardos made a similar observation suggesting that the nice structural property fails for MIN-MAX-MULTIWAY-CUT, i.e., for $p = \infty$ – in contrast, our example in Figure 1 shows that the nice structural property fails to hold for every $p > 1$.

We now discuss our hardness results for ℓ_p -NORM-MULTIWAY-CUT.

► **Theorem 1.** *We have the following hardness results for ℓ_p -NORM-MULTIWAY-CUT.*

1. ℓ_p -NORM-MULTIWAY-CUT is NP-hard for every $p > 1$ and every $k \geq 4$.
2. ℓ_p -NORM-MULTIWAY-CUT in planar graphs is NP-hard for every $p > 1$.

We note that the case of $p = 1$ and $p = \infty$ are already known to be hard: MULTIWAY-CUT is NP-hard for $k = 3$ terminals and is NP-hard in planar graphs when k is arbitrary (i.e., when k is not a fixed constant) [14]; MIN-MAX-MULTIWAY-CUT is NP-hard for $k = 4$ terminals and is NP-hard in trees when k is arbitrary [22]. Our NP-hardness in planar graphs result also requires k to be arbitrary.



■ **Figure 1** An example where the unique optimum partition for ℓ_p -NORM-MULTIWAY-CUT for $k = 5$ induces a disconnected part for every $p > 1$. The edge weights are as shown with $a := 8^{p/(p-1)}$ and the set of terminals is $\{u_1, v_1, v_2, v_3, v_4\}$. A partition that puts u_2 with one of the terminals in $\{v_1, v_2, v_3, v_4\}$ (and isolates the remaining terminals) has ℓ_p -norm objective value $((3a + 3)^p + 3(3a + 2)^p + 4^p)^{1/p}$ and the partition that puts u_2 with u_1 (and isolates the remaining terminals) has ℓ_p -norm objective value $(4(3a + 2)^p + 8^p)^{1/p}$ – the latter is strictly cheaper by the choice of a .

Given that the problem is NP-hard, we focus on designing approximation algorithms. We show the following result:

► **Theorem 2.** *There exists a polynomial-time $O(\log^{1.5} n \log^{0.5} k)$ -approximation for ℓ_p -NORM-MULTIWAY-CUT for every $p \geq 1$, where n is the number of vertices and k is the number of terminals in the input instance.*

We note that our approximation factor is $O(\log^2 n)$ since $k \leq n$. While it might be tempting to design an approximation algorithm by solving a convex programming relaxation for ℓ_p -NORM-MULTIWAY-CUT and rounding it, we rule out this approach: the natural convex programming relaxation has an integrality gap of $\Omega(k^{1-1/p})$ – see Section 4. Hence, our approach for the approximation algorithm is not based on a convex program but instead based on combinatorial techniques.

For comparison, we state the currently best known approximation factors for $p = 1$ and $p = \infty$: MULTIWAY-CUT admits a 1.2965-approximation via an LP-based algorithm [21] and MIN-MAX-MULTIWAY-CUT admits an $O(\sqrt{\log n \log k})$ -approximation based on a bicriteria approximation for the small-set expansion problem [3].

As a final result, we show that removing the dependence on the number n of vertices in the approximation factor of ℓ_p -NORM-MULTIWAY-CUT is hard assuming the small set expansion hypothesis [20]. In particular, we show that achieving a $(k^{1-1/p-\epsilon})$ -approximation for any constant $\epsilon > 0$ is hard. We note that there is a trivial $O(k^{1-1/p})$ -approximation for ℓ_p -NORM-MULTIWAY-CUT.

1.2 Outline of techniques

We briefly outline the techniques underlying our results.

Hardness results

We show hardness of ℓ_p -NORM-MULTIWAY-CUT for $k = 4$ terminals by a reduction from the graph bisection problem. Our main tool to control the ℓ_p -norm objective in our hardness reduction is the Mean Value Theorem and its consequences. In order to show NP-hardness of ℓ_p -NORM-MULTIWAY-CUT in planar graphs, we reduce from the 3-partition problem. We do a gadget based reduction where the gadget is planar. We note that the number of terminals in this reduction is not a constant and is $\Omega(n)$, where n is the number of vertices. Once again, we rely on the Mean Value Theorem and its consequences to control the ℓ_p -norm objective in the reduction. We mention that the starting problems in our hardness reductions are inspired by the hardness results shown by Svitkina and Tardos for MIN-MAX-MULTIWAY-CUT: they showed that MIN-MAX-MULTIWAY-CUT is NP-hard for $k = 4$ terminals by a reduction from the graph bisection problem and that MIN-MAX-MULTIWAY-CUT is NP-hard in trees by a reduction from the 3-partition problem. We also use these same starting problems, but our reductions are more involved owing to the ℓ_p -norm nature of the objective.

Approximation algorithm

For the purposes of the algorithm, we will assume knowledge of the optimum value, say OPT – such a value can be guessed within a factor of 2 via binary search. Our approximation algorithm proceeds in three steps. We describe these three steps now.

In the first step of the algorithm, we obtain a collection \mathcal{S} of subsets of the vertex set satisfying four properties: (1) each set S in the collection \mathcal{S} has at most one terminal, (2) the ℓ_p -norm of the cut values of the sets in the collection raised to the p th power is small, i.e., $\sum_{S \in \mathcal{S}} w(\delta(S))^p = (\beta^p \log n) \text{OPT}^p$ where $\beta = O(\sqrt{\log n \log k})$, (3) the number of sets in the collection \mathcal{S} is $O(k \log n)$, and (4) the union of the sets in the collection \mathcal{S} is V . We perform this first step via a multiplicative updates method. For this, we use a bicriteria approximation algorithm for the unbalanced terminal cut problem which was given by Bansal et al [3] (see Section 2 for a description of the unbalanced terminal cut problem and the bicriteria approximation).

Although property (2) gives a bound on the ℓ_p -norm of the cut values of the sets in the collection \mathcal{S} relative to the optimum, the collection \mathcal{S} does not correspond to a feasible multiway cut: recall that a feasible multiway cut is a partition $\mathcal{P} = (P_1, \dots, P_k)$ of the vertex set where each P_i contains exactly one terminal. The objective of the next two steps is to refine the collection \mathcal{S} to achieve feasibility without blowing up the ℓ_p -norm of the cut values of the parts.

In the second step of the algorithm, we uncross the sets in the collection \mathcal{S} to obtain a partition \mathcal{Q} without increasing the cut values of the sets. We crucially exploit the posimodularity property of the graph cut function to achieve this: posimodularity implies that for all subsets $A, B \subseteq V$ of vertices, either $w(\delta(A)) \geq w(\delta(A - B))$ or $w(\delta(B)) \geq w(\delta(B - A))$. We iteratively consider all pairs of crossing subsets A, B in the collection \mathcal{S} and replace A with $A - B$ if $w(\delta(A)) \geq w(\delta(A - B))$ or replace B with $B - A$ if $w(\delta(B)) \geq w(\delta(B - A))$. The outcome of this step is a partition \mathcal{Q} of the vertex set V satisfying three properties: (i) each part Q in the partition \mathcal{Q} has at most one terminal, (ii) the ℓ_p -norm of the cut values of the parts in the partition \mathcal{Q} raised to the p th power is still small, i.e., $\sum_{Q \in \mathcal{Q}} w(\delta(Q))^p = (\beta^p \log n) \text{OPT}^p$, and (iii) the number of parts in the partition \mathcal{Q} is $O(k \log n)$.

Once again, we observe that the partition \mathcal{Q} at the end of the second step may not correspond to a feasible multiway cut: we could have more than k parts in \mathcal{Q} with some of the parts having no terminals. We address this issue in the third step by a careful aggregation.

For the third step of the algorithm, let Q_i be the part in \mathcal{Q} that contains terminal t_i – we have k such parts by property (i) – and let R_1, \dots, R_t be the remaining parts in \mathcal{Q} that contain no terminals. We will aggregate the remaining parts of \mathcal{Q} into the k parts Q_1, \dots, Q_k without blowing up the ℓ_p -norm of the cut value of the parts. By property (iii), the number of remaining parts t is $O(k \log n)$. We create k disjoint buckets B_1, \dots, B_k where B_i contains the union of $O(\log n)$ many parts among R_1, \dots, R_t . Finally, we merge B_i with Q_i . This results in a partition $\mathcal{P} = (Q_1 \cup B_1, \dots, Q_k \cup B_k)$ of V with terminal t_i being in the i th part $Q_i \cup B_i$. The key now is to control the blow-up in the p th power of the ℓ_p -norm of the cut values of the parts in \mathcal{P} : we bound this by a $O(\log^{p-1} n)$ -factor relative to the p th power of the ℓ_p -norm of the cut values of the parts in \mathcal{Q} via Jensen's inequality; while using Jensen's inequality, we exploit the fact that each bucket contained $O(\log n)$ many parts. Consequently, using property (ii), the ℓ_p -norm objective value of the cut values of the parts in the partition \mathcal{P} raised to the p th power is still small – we show that $\sum_{P \in \mathcal{P}} w(\delta(P))^p = \beta^p \log^p n \text{OPT}^p$ and hence, we have an approximation factor of $O(\beta \log n)$.

The first step of our algorithm is inspired by the $O(\log n)$ -approximation algorithm for MIN-MAX-MULTIWAY-CUT due to Bansal et al [3] – we modify the multiplicative weights update method and adapt it for ℓ_p -NORM-MULTIWAY-CUT. Our second and third steps differ from that of Bansal et al. We mention that the second and third steps of our algorithm can be adapted to achieve an $O(\beta \log n)$ -approximation factor for ℓ_p -NORM-MULTIWAY-CUT for $p = \infty$, but the resulting approximation factor is only $O(\log^2 n)$ which is weaker than the $O(\log n)$ -factor achieved by Bansal et al. The additional loss of $\log n$ -factor in our algorithm comes from the third step (i.e., the aggregation step). The aggregation step designed in [3] is randomized and saves the $\log n$ -factor in expectation, but it does not generalize to ℓ_p -NORM-MULTIWAY-CUT. As mentioned before, the second step of our algorithm relies on posimodularity. The posimodularity property of the graph cut function has been used in previous works for MIN-MAX-MULTIWAY-CUT in an implicit fashion by a careful and somewhat tedious edge counting argument [3, 22]. We circumvent the edge counting argument here by the clean posimodularity abstraction. Moreover, the posimodularity abstraction makes the counting considerably easier for our more general problem of ℓ_p -NORM-MULTIWAY-CUT.

1.3 Related Work

ℓ_p -NORM-MULTIWAY-CUT can be viewed as a fairness inducing objective in the context of multiway partitioning problems. Recent works have proposed and studied various fairness inducing objectives for graph cuts and partitioning that are different from ℓ_p -NORM-MULTIWAY-CUT. We briefly discuss these works here. All of the works mentioned in this subsection consider a more general problem known as *correlation clustering* – we discuss these works by specializing to cut and partitioning problems since these specializations are the ones related to our work.

Puleo and Milenkovic [19] introduced a local vertex-wise min-max objective for min (s, t) -cut – here, the goal is to partition the vertex set V of the given edge-weighted undirected graph into two parts $(S, V \setminus S)$ each containing exactly one of the terminals in $\{s, t\}$ so as to minimize $\max_{v \in V} w(\delta(v) \cap \delta(S))$. The motivation behind this objective is that the cut should be fair to every vertex in the graph, i.e., no vertex should pay a lot for the edges in the cut. A result of Chvátal [13] implies that this problem is $(2 - \epsilon)$ -inapproximable for every constant $\epsilon > 0$. Charikar, Gupta, and Schwartz [10] gave an $O(\sqrt{n})$ -approximation for this problem. Reducing the approximability vs inapproximability gap for this problem remains an intriguing open problem. Kalhan, Makarychev, and Zhou [15] considered an ℓ_p -norm

version of the objective where the goal is to minimize $(\sum_{v \in V} w(\delta(v) \cap \delta(S))^p)^{1/p}$ and gave an $O(n^{\frac{1}{2} - \frac{1}{2p}} \log^{\frac{1}{2} - \frac{1}{2p}} n)$ -approximation, thus interpolating the best known results for $p = 1$ and $p = \infty$.

Ahmadi, Khuller, and Saha [1] introduced a min-max version of multicut: the input consists of an undirected graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{R}_+$ along with source-sink terminal pairs $(s_1, t_1), \dots, (s_k, t_k)$. The goal is to find a partition $\mathcal{P} = (P_1, \dots, P_r)$ of the vertex set with all source-sink pairs separated by the partition so as to minimize $\max_{i \in [r]} w(\delta(P_i))$. We emphasize that the number of parts here – namely, r – is not constrained by the input and hence, could be arbitrary. Ahmadi, Khuller, and Saha gave an $O(\sqrt{\log n \max\{\log k, \log T\}})$ -approximation for this problem, where T is the number of parts in the optimal solution. Kalhan, Makarychev, and Zhou [15] improved the approximation factor to $2 + \epsilon$.

Organization

We begin with preliminaries in Section 2. We present the complete details of our approximation algorithm and prove Theorem 2 in Section 3. We discuss a convex program and its integrality gap in Section 4. Due to space limitations, the proofs of the rest of the results appear in the complete version of this work [9]. We conclude with a few open problems in Section 5.

2 Preliminaries

We start with notations that will be used throughout. Let $G = (V, E)$ be an undirected graph with edge weight function $w : E \rightarrow \mathbb{R}_+$ and vertex weight function $y : V \rightarrow \mathbb{R}_+$. For every subset $S \subseteq V$, we use $\delta_G(S)$ to denote the set of edges that have exactly one end-vertex in S (we will drop the subscript G when the graph is clear from context), and we write $w(\delta(S)) := \sum_{e \in \delta(S)} w(e)$. Moreover, we will use $y(S)$ to refer to $\sum_{v \in S} y(v)$. We will denote an instance of ℓ_p -NORM-MULTIWAY-CUT by (G, w, T) , where $G = (V, E)$ is the input graph, $w : E \rightarrow \mathbb{R}_+$ is the edge weight function, and $T \subseteq V$ is the set of terminal vertices. We will call a partition $\tilde{\mathcal{P}} = (P_1, \dots, P_r)$ of the vertex set to be a multiway cut if $r = k$ and $t_i \in P_i$ for each $i \in [k]$ and denote the ℓ_p -norm of the cut values of the parts (i.e., $(\sum_{i=1}^k w(\delta(P_i))^p)^{1/p}$) as the ℓ_p -norm objective value of the multiway cut $\tilde{\mathcal{P}}$.

We note that the function $\mu : \mathbb{R} \rightarrow \mathbb{R}$ defined by $\mu(x) := x^p$ is convex for every $p \geq 1$. We will use Jensen's inequality as stated below in our approximation algorithm as well as our hardness reductions.

► **Lemma 3** (Jensen). *Let $\mu : \mathbb{R} \rightarrow \mathbb{R}$ be a convex function. For arbitrary $x_1, \dots, x_t \in \mathbb{R}$, we have*

$$\mu\left(\frac{1}{t} \sum_{i=1}^t x_i\right) \leq \frac{1}{t} \sum_{i=1}^t \mu(x_i).$$

Our algorithm relies on the graph cut function being symmetric and submodular. We recall that the graph cut function $f : 2^V \rightarrow \mathbb{R}_+$ is given by $f(S) := w(\delta(S))$ for all $S \subseteq V$. Let $f : 2^V \rightarrow \mathbb{R}_+$ be a set function. The function f is symmetric if $f(S) = f(V \setminus S)$ for all $S \subseteq V$, submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all $A, B \subseteq V$, and posimodular if $f(A) + f(B) \geq f(A - B) + f(B - A)$ for all $A, B \subseteq V$. Symmetric submodular functions are also posimodular (see Proposition 4) – this fact has been used implicitly [3, 22] and explicitly [8, 11] before.

► **Proposition 4.** *Symmetric submodular functions are posimodular.*

Proof. Let $f : 2^V \rightarrow \mathbb{R}$ be a symmetric submodular set function on a set V , and let $A, B \subseteq V$ be two arbitrary subsets. Then, we have

$$\begin{aligned} f(A) + f(B) &= f(V - A) + f(B) \geq f((V - A) \cup B) + f((V - A) \cap B) \\ &= f(V - (A - B)) + f(B - A) = f(A - B) + f(B - A). \end{aligned}$$

In the above, the first and last equations follow by symmetry and the inequality follows by submodularity. ◀

Our algorithm for ℓ_p -NORM-MULTIWAY-CUT relies on an intermediate problem, namely the Unbalanced Terminal Cut problem that we introduce now. In Unbalanced Terminal Cut (UTC), the input (G, w, y, τ, T) consists of an undirected graph $G = (V, E)$, an edge weight function $w : E \rightarrow \mathbb{R}_+$, a vertex weight function $y : V \rightarrow \mathbb{R}_+$, a real value $\tau \in [0, 1]$, and a set $T \subseteq V$ of terminal vertices. The goal is to compute

$$\text{UTC}(G, w, y, \tau, T) := \min \{w(\delta(S)) : S \subseteq V, y(S) \geq \tau \cdot y(V), |S \cap T| \leq 1\}.$$

Bansal et al. gave a bicriteria approximation for UTC that is summarized in the theorem below.

► **Theorem 5.** [3] *There exists an algorithm UTC-BICRIT-ALGO that takes as input (G, w, y, τ, T) consisting of an undirected graph $G = (V, E)$, an edge weight function $w : E \rightarrow \mathbb{R}_+$, a vertex weight function $y : V \rightarrow \mathbb{R}_+$, a number $\tau \in [0, 1]$, and a set $T \subseteq V$ of terminal vertices and runs in polynomial time to return a set $S \subseteq V$ such that*

1. $|S \cap T| \leq 1$,
2. $y(S) = \Omega(\tau)y(V)$, and
3. $w(\delta(S)) \leq \alpha \text{UTC}(G, w, y, \tau, T)$, where $\alpha = O(\sqrt{\log n \log(1/\tau)})$ and $n = |V|$.

3 Approximation Algorithm

Let OPT be the optimal ℓ_p -norm objective value of a multiway cut in the given instance. For the purposes of the algorithm, we will assume knowledge of a value D such that $D \geq \text{OPT}^p$ – such a value can be guessed via binary search.

Our approximation algorithm to prove Theorem 2 involves three steps. In the first step of the algorithm, we will obtain a collection \mathcal{S} of $O(k \log n)$ sets whose union is the vertex set V such that each set in the collection has at most one terminal, the cut value of each set is not too large relative to D , and the ℓ_p -norm of the cut values of the sets in the collection is within a $\text{polylog}(n)$ factor of D (see Lemma 6). Although the collection \mathcal{S} has low ℓ_p -norm value relative to D , the collection \mathcal{S} may not be a feasible multiway cut. In the second step of the algorithm, we uncross the sets in the collection \mathcal{S} without increasing the ℓ_p -norm of the cut values of the sets in the collection (see Lemma 10). After uncrossing, we obtain a partition, but we could have more than k sets. We address this in our third step, where we aggregate parts to ensure that we obtain exactly k parts (see Lemma 11). We rely on Jensen's inequality to ensure that the aggregation does not blow-up the ℓ_p -norm of the cut values of the sets in the partition.

We begin with the first step of the algorithm in Lemma 6.

► **Lemma 6.** *There exists an algorithm that takes as input an undirected graph $G = (V, E)$, an edge weight function $w : E \rightarrow \mathbb{R}_+$, k distinct terminal vertices $T := \{t_1, \dots, t_k\} \subseteq V$ and a value $D > 0$ such that there exists a partition (P_1^*, \dots, P_k^*) of V with $t_i \in P_i^*$ for all $i \in [k]$ and $\sum_{i=1}^k w(\delta(P_i^*))^p \leq D$, and runs in polynomial time to return a collection of sets $\mathcal{S} \subseteq 2^V$ that satisfies the following:*

29:8 ℓ_p -Norm Multiway Cut

1. $|S \cap T| \leq 1$ and $w(\delta(S)) \leq \beta(2D)^{1/p}$ for every $S \in \mathcal{S}$,
 2. $\sum_{S \in \mathcal{S}} w(\delta(S))^p = \beta^p(\log n)D$, and
 3. $|\mathcal{S}| = O(k \log n)$ and $|\{S \in \mathcal{S} : v \in S\}| \geq \log n$ for each $v \in V$,
- where $\beta = O(\sqrt{\log n \log k})$.

Proof. We will use Algorithm 1 to obtain the desired collection \mathcal{S} . We will show the correctness of Algorithm 1 based on Claims 7, 8 and 9.

■ **Algorithm 1** Multiplicative weights update.

```

Initialize  $t \leftarrow 1$ ,  $\mathcal{S} \leftarrow \emptyset$ ,  $y^1(v) = 1$  for each  $v \in V$ ,  $Y^1 = \sum_{v \in V} y^1(v)$  and
 $\beta = O(\sqrt{\log n \log k})$ 
while  $Y^t > \frac{1}{n}$  do
  for  $i = 1, 2, \dots, \log(2k)$  do
    Execute UTC-BICRIT-ALGO( $G, w, y^t, 2^{-i}, T$ ) to obtain a subset  $S^t(i) \subseteq V$ 
    if  $w(\delta(S^t(i))) \leq \beta(\frac{4D}{2^i})^{1/p}$  then
      Set  $S^t = S^t(i)$  and BREAK
    end if
  end for
   $\mathcal{S} \leftarrow \mathcal{S} \cup \{S^t\}$ .
  for  $v \in V$  do
    Set  $y^{t+1}(v) = \begin{cases} y^t(v)/2 & \text{if } v \in S^t, \\ y^t(v) & \text{if } v \in V \setminus S^t. \end{cases}$ 
  end for
  Set  $Y^{t+1} = \sum_{v \in V} y^{t+1}(v)$ .
  Set  $t \leftarrow t + 1$ .
end while
Return  $\mathcal{S}$ 

```

Our first claim will help in showing that the set S^t added in each iteration of the while loop satisfies certain nice properties.

▷ **Claim 7.** For every iteration t of the while loop of Algorithm 1, there exists $i \in \{1, 2, \dots, \log(2k)\}$ such that the set $S^t(i)$ satisfies the following conditions:

1. $|S^t(i) \cap T| \leq 1$,
2. $y^t(S^t(i)) = \Omega(\frac{Y^t}{2^i})$, and
3. $w(\delta(S^t(i))) \leq \beta(\frac{4D}{2^i})^{1/p}$.

Proof. We have that $\sum_{i=1}^k y^t(P_i^*) = y^t(V)$ and

$$\sum_{i=1}^k w(\delta(P_i^*))^p \leq D.$$

Let L be the subset of indices of parts for which the cut value is relatively low:

$$L := \left\{ j \in [k] : w(\delta(P_j^*))^p \leq \frac{2y^t(P_j^*)}{Y^t} \cdot D \right\}.$$

It follows that

$$\sum_{j \in [k] \setminus L} y^t(P_j^*) < \sum_{j \in [k] \setminus L} \frac{w(\delta(P_j^*))^p Y^t}{2D} \leq \frac{Y^t}{2}$$

and hence,

$$\sum_{j \in L} y^t(P_j^*) = Y^t - \sum_{j \in [k] \setminus L} y^t(P_j^*) > Y^t - \frac{Y^t}{2} = \frac{Y^t}{2}.$$

Since $|L| \leq k$, there exists an index $q \in L$ such that $y^t(P_q^*) > Y^t/(2k)$. Let us fix i_0 to be an integer such that $y^t(P_q^*) \in (Y^t \cdot 2^{-i_0}, Y^t \cdot 2^{-i_0+1}]$. Then, we must have $i_0 \leq \log(2k)$. We note that the set P_q^* satisfies $|P_q^* \cap T| = 1$ and $y^t(P_q^*) > Y^t/(2k) = y^t(V)/(2k)$. This implies P_q^* is feasible to the UTC problem on input $(G, w, y^t, 1/2^{i_0}, T)$. Therefore, according to Theorem 5, the set $S^t(i_0)$ has the following properties: Firstly, $|S^t(i_0) \cap T| \leq 1$. Secondly, $y^t(S^t(i_0)) = \Omega(1/2^{i_0})y^t(V) = \Omega(Y^t/2^{i_0})$. Finally,

$$\begin{aligned} w(\delta(S^t(i_0))) &= O(\sqrt{\log n \log(2k)}) \cdot \text{UTC}\left(G, w, y^t, \frac{1}{2^{i_0}}, T\right) \\ &= O(\sqrt{\log n \log k}) \cdot w(\delta(P_q^*)) \\ &= O(\sqrt{\log n \log k}) \cdot \left(\frac{2y^t(P_q^*)}{Y^t} \cdot D\right)^{\frac{1}{p}} \\ &= O(\sqrt{\log n \log k}) \cdot \left(\frac{2 \cdot Y^t \cdot 2^{-i_0+1}}{Y^t} \cdot D\right)^{\frac{1}{p}} \\ &= O(\sqrt{\log n \log k}) \cdot \left(\frac{4D}{2^{i_0}}\right)^{\frac{1}{p}}. \end{aligned}$$

This completes the proof of Claim 7. \triangleleft

For the rest of the proof, we will use the following notation: In the t 'th iteration of the while loop of Algorithm 1, we will fix $i_t \in \{1, 2, \dots, \log(2k)\}$ to be the integer such that $S^t = S^t(i_t)$. We will use ℓ to denote the total number of iterations of the while loop. For each $v \in V$, We define $N_v := |\{t \in [\ell] : v \in S^t\}|$ to be the number of sets in the collection \mathcal{S} that contain the vertex v .

We observe that for each $v \in V$, we have $y^{\ell+1}(v) = 2^{-N_v}$. Claim 7 and Theorem 5 together imply that the t 'th iteration of the while loop leads to a set S^t being added to the collection \mathcal{S} such that

1. $|S^t \cap T| \leq 1$,
2. $y^t(S^t) = \Omega(\frac{Y^t}{2^{i_t}})$, and
3. $w(\delta(S^t)) \leq \beta(\frac{4D}{2^{i_t}})^{1/p}$.

Our next claim shows that the number of iterations of the while loop executed in Algorithm 1 is small. Moreover, the union of the sets in the collection \mathcal{S} is the vertex set V .

\triangleright **Claim 8.** The number of iterations ℓ of the while loop satisfies $\ell = O(k \log n)$. Moreover, $N_v \geq \log n$ for each $v \in V$.

Proof. Upon termination of Algorithm 1, we must have $Y^{\ell+1} \leq 1/n$. Combining with the earlier observation that $y^{\ell+1}(v) = 2^{-N_v}$ for every $v \in V$, we have that

$$2^{-N_v} = y^{\ell+1}(v) \leq Y^{\ell+1} \leq \frac{1}{n},$$

which implies that $N_v \geq \log n$ for every $v \in V$.

29:10 ℓ_p -Norm Multiway Cut

It remains to show that $\ell = O(k \log n)$. Consider the t th iteration of the while loop for an arbitrary $t \in [\ell]$. By property 2 of the set S^t stated above, we have that $y^t(S^t) \geq cY^t/2^{i_t} \geq cY^t/(2k)$ for some constant $c > 0$. Consequently,

$$Y^{t+1} = Y^t - \frac{y^t(S^t)}{2} \leq Y^t - \frac{cY^t}{4k} = \left(1 - \frac{c}{4k}\right) Y^t.$$

Due to the termination condition of the while loop, we know that $Y^\ell > 1/n$. Hence,

$$\frac{1}{n} < Y^\ell \leq \left(1 - \frac{c}{4k}\right)^{\ell-1} Y^1 = \left(1 - \frac{c}{4k}\right)^{\ell-1} n \leq \exp\left(-\frac{c(\ell-1)}{4k}\right) n.$$

Therefore, $\frac{c(\ell-1)}{4k} = O(\log n)$ which implies that $\ell = O(k \log n)$. This completes the proof of Claim 8. \triangleleft

The next claim bounds the ℓ_p -norm of the cut values of the sets in the collection \mathcal{S} .

\triangleright **Claim 9.** The collection \mathcal{S} returned by Algorithm 1 satisfies $\sum_{S \in \mathcal{S}} w(\delta(S))^p = O(\beta^p \log n) \cdot D$.

Proof. Consider the t th iteration of the while loop for an arbitrary $t \in [\ell]$. By property 3 of the set S^t stated above, we have that $w(\delta(S^t)) \leq \beta(4D/2^{i_t})^{1/p}$ and consequently, $2^{i_t} \leq 4D\beta^p \cdot w(\delta(S^t))^{-p}$. Moreover, by property 2 of the set S^t stated above, we have that $y^t(S^t) \geq cY^t/2^{i_t}$ for some constant $c > 0$. Hence,

$$y^t(S^t) \geq \frac{cY^t}{2^{i_t}} \geq \frac{cY^t \cdot w(\delta(S^t))^p}{\beta^p \cdot 4D}.$$

Therefore,

$$Y^{t+1} = Y^t - \frac{y^t(S^t)}{2} \leq \left(1 - \frac{c \cdot w(\delta(S^t))^p}{\beta^p \cdot 8D}\right) Y^t.$$

Using the fact that $Y^\ell > 1/n$, we observe that

$$\begin{aligned} \frac{1}{n} < Y^\ell &\leq Y^1 \cdot \prod_{t=1}^{\ell-1} \left(1 - \frac{c \cdot w(\delta(S^t))^p}{\beta^p \cdot 8D}\right) = n \cdot \prod_{t=1}^{\ell-1} \left(1 - \frac{c \cdot w(\delta(S^t))^p}{\beta^p \cdot 8D}\right) \\ &\leq n \cdot \prod_{i=1}^{\ell-1} \exp\left(-\frac{c \cdot w(\delta(S^t))^p}{\beta^p \cdot 8D}\right) = n \cdot \exp\left(-\frac{c \cdot \sum_{i=1}^{\ell-1} w(\delta(S^t))^p}{\beta^p \cdot 8D}\right). \end{aligned}$$

This implies that $\frac{c \cdot \sum_{i=1}^{\ell-1} w(\delta(S^t))^p}{\beta^p \cdot 8D} = O(\log n)$, and hence $\sum_{i=1}^{\ell-1} w(\delta(S^t))^p = O(\beta^p \log n) \cdot D$.

In the ℓ 'th iteration of the while loop, we have $w(\delta(S^\ell)) \leq \beta(4D/2^{i_\ell})^{1/p}$ by property 3 of the set S^t stated above and hence $w(\delta(S^\ell))^p \leq \beta^p \cdot 4D/2^{i_\ell} \leq O(\beta^p D)$. Consequently, $\sum_{i=1}^{\ell} w(\delta(S^t))^p = O(\beta^p \log n) \cdot D$. This completes the proof of Claim 9. \triangleleft

We now show correctness of our algorithm to complete the proof of Lemma 6. Firstly, we note that every $S \in \mathcal{S}$ satisfies $|S \cap T| \leq 1$ by property 1 of the set S^t stated above. Moreover, we have $w(\delta(S)) \leq \beta(4D/2^{i_t})^{1/p} \leq \beta(2D)^{1/p}$, which implies conclusion 1 in Lemma 6. Secondly, Conclusion 2 in Lemma 6 is implied by Claim 9. Finally, conclusion 3 of Lemma 6 is implied by Claim 8 because each iteration of the while loop adds exactly one new set to the collection \mathcal{S} .

We now bound the run time of Algorithm 1. Each iteration of the while loop takes polynomial time due to Theorem 5, and the number of iterations of the while loop is $O(k \log n)$. This implies that the total run time of Algorithm 1 is indeed polynomial in the size of the input. \blacktriangleleft

The collection \mathcal{S} that we obtain in Lemma 6 may not be a partition. Our next lemma will uncross the collection \mathcal{S} obtained from Lemma 6 to obtain a partition without increasing the cut values of the sets.

► **Lemma 10.** *There exists an algorithm that takes as input a collection $\mathcal{S} \subseteq 2^V$ of subsets of vertices satisfying the conclusions in Lemma 6 and runs in polynomial time to return a partition $\tilde{\mathcal{Q}}$ of V such that*

1. $|Q \cap T| \leq 1$ for each $Q \in \tilde{\mathcal{Q}}$,
2. $\sum_{Q \in \tilde{\mathcal{Q}}} w(\delta(Q))^p \leq \sum_{S \in \mathcal{S}} w(\delta(S))^p$, and
3. the number of parts in $\tilde{\mathcal{Q}}$ is $O(k \log n)$.

Proof. For convenience, we will define $f : 2^V \rightarrow \mathbb{R}_+$ by $f(S) := w(\delta(S))$ for all $S \subseteq V$. We will use Algorithm 2 to obtain the desired partition $\tilde{\mathcal{Q}}$ of V .

■ **Algorithm 2** Uncrossing.

```

Initialize  $\tilde{\mathcal{Q}} \leftarrow \mathcal{S}$ 
while there exist distinct sets  $A, B \in \tilde{\mathcal{Q}}$  such that  $A \cap B \neq \emptyset$  do
    if  $f(A) \geq f(A - B)$  then
        Set  $A \leftarrow A - B$ 
    else
        Set  $B \leftarrow B - A$ 
    end if
end while
Return  $\tilde{\mathcal{Q}}$ 

```

We now prove the correctness of Algorithm 2. We begin by observing that Algorithm 2 indeed outputs a partition of the vertex set: Firstly, the while loop enforces that the output $\tilde{\mathcal{Q}}$ satisfies $A \cap B = \emptyset$ for all distinct $A, B \in \tilde{\mathcal{Q}}$. Secondly, during each iteration of the while loop, the set $\bigcup_{Q \in \tilde{\mathcal{Q}}} Q$ remains unchanged: In the iteration of the while loop that uncrosses $A, B \in \tilde{\mathcal{Q}}$, let A' and B' denote the updated sets at the end of the while loop, respectively. Then we must have $A' \cup B' = (A - B) \cup B = A \cup B$ or $A' \cup B' = A \cup (B - A) = A \cup B$. In either case, since $A' \cup B' = A \cup B$, the set $\bigcup_{Q \in \tilde{\mathcal{Q}}} Q$ remains unchanged after the update. Therefore, we have $\bigcup_{Q \in \tilde{\mathcal{Q}}} Q = \bigcup_{S \in \mathcal{S}} S$. We recall that $\bigcup_{S \in \mathcal{S}} S = V$ by conclusion 3 of Lemma 6. Hence, $\tilde{\mathcal{Q}}$ is indeed a partition of V .

Furthermore, each set Q in the output $\tilde{\mathcal{Q}}$ is a subset of some set $S \in \mathcal{S}$. This implies $|Q \cap T| \leq |S \cap T| \leq 1$, thus proving the first conclusion.

To prove the second conclusion, we use posimodularity of f as shown in Proposition 4. Namely, for every $A, B \subseteq V$,

$$f(A) + f(B) \geq f(A - B) + f(B - A).$$

Therefore, at least one of the following two hold: either $f(A) \geq f(A - B)$ or $f(B) \geq f(B - A)$. This implies that, by the choice of the algorithm, $\sum_{Q \in \tilde{\mathcal{Q}}} f(Q)^p$ does not increase.

To see the third conclusion, we note that after each iteration of the while loop, the size of $\tilde{\mathcal{Q}}$ is unchanged. Therefore, at the end Algorithm 2, we have $|\tilde{\mathcal{Q}}| = |\mathcal{S}| = O(k \log n)$ by Lemma 6.

Finally, we bound the run time as follows. At initialization, there are $O((k \log n)^2)$ pairs $(A, B) \in \tilde{\mathcal{Q}}^2$ such that $A \cap B \neq \emptyset$. After each iteration of the while loop, the number of such pairs decreases by at least 1. Therefore, the total number of iterations of the while loop is $O((k \log n)^2)$. Hence, Algorithm 2 indeed runs in polynomial time. ◀

The partition $\tilde{\mathcal{Q}}$ that we obtain in Lemma 10 may contain more than k parts and hence, some of the parts may not contain any terminals. Our next lemma will aggregate the parts in $\tilde{\mathcal{Q}}$ from Lemma 10 to obtain a k -partition that contains exactly one terminal in each part while controlling the increase in the ℓ_p -norm of the cut value of the parts.

► **Lemma 11.** *There exists an algorithm that takes as input a partition $\tilde{\mathcal{Q}}$ of V satisfying the conclusions in Lemma 10 and runs in polynomial time to return a partition (P_1, P_2, \dots, P_k) of V such that*

1. $t_i \in P_i$ for each $i \in [k]$, and
2. $\sum_{i=1}^k w(\delta(P_i))^p = O((\beta \log n)^p) \cdot D$.

Proof. We will use Algorithm 3 on input $\tilde{\mathcal{P}}$ to obtain the desired partition.

■ **Algorithm 3** Aggregating.

Let $\mathcal{F} = \{Q \in \tilde{\mathcal{Q}} : Q \cap T = \emptyset\}$.

Let $\mathcal{P}' = \{Q \in \tilde{\mathcal{Q}} : Q \cap T \neq \emptyset\} = \{Q'_1, \dots, Q'_k\}$, where $t_i \in Q'_i$ for each $i \in [k]$.

Partition the sets in \mathcal{F} into k buckets B_1, \dots, B_k such that $|B_i| = O(\log n)$ for each $i \in [k]$ (arbitrarily).

for $i = 1, 2, \dots, k$ **do**

Set $P_i \leftarrow Q'_i \cup (\bigcup_{A \in B_i} A)$

end for

Return (P_1, \dots, P_k) .

The run time of Algorithm 3 is linear in its input size. We now argue the correctness. We note that the third step in Algorithm 3 is possible because $|\mathcal{F}| \leq |\tilde{\mathcal{Q}}| = O(k \log n)$.

Since $|Q \cap T| \leq 1$ for each $Q \in \tilde{\mathcal{Q}}$, the tuple (P_1, \dots, P_k) returned by Algorithm 3 is indeed a partition of V satisfying $t_i \in P_i$ for all $i \in [k]$. We will now bound $\sum_{i=1}^k f(P_i)^p$, where $f : 2^V \rightarrow \mathbb{R}_+$ is given by $f(S) := w(\delta(S))$ for all $S \subseteq V$. We have that

$$\sum_{i=1}^k f(P_i)^p = \sum_{i=1}^k f\left(Q'_i \cup \left(\bigcup_{A \in B_i} A\right)\right)^p \leq \sum_{i=1}^k \left(f(Q'_i) + \sum_{A \in B_i} f(A)\right)^p.$$

Since the number of sets in B_i is $O(\log n)$, we have the following using Jensen's inequality (Lemma 3) for each $i \in [k]$:

$$\begin{aligned} \left(f(Q'_i) + \sum_{A \in B_i} f(A)\right)^p &\leq (|B_i| + 1)^{p-1} \left(f(Q'_i)^p + \sum_{A \in B_i} f(A)^p\right) \\ &= O(\log^{p-1} n) \left(f(Q'_i)^p + \sum_{A \in B_i} f(A)^p\right). \end{aligned}$$

Hence,

$$\begin{aligned} \sum_{i=1}^k f(P_i)^p &= \sum_{i=1}^k O(\log^{p-1} n) \left(f(Q'_i)^p + \sum_{A \in B_i} f(A)^p\right) = O(\log^{p-1} n) \sum_{Q \in \tilde{\mathcal{Q}}} f(Q)^p \\ &= O(\log^{p-1} n) \sum_{S \in \mathcal{S}} f(S)^p = \beta^p O(\log^p n) D. \end{aligned}$$

The last but one equality above is due to conclusion 2 of Lemma 10bbb and the last equality is due to conclusion 2 of Lemma 6. Hence, $\sum_{i=1}^k w(\delta(P_i))^p = \sum_{i=1}^k f(P_i)^p = O((\beta \log n)^p) D$. ◀

Lemmas 6, 10, and 11 together lead to an algorithm that takes as input an undirected graph $G = (V, E)$, an edge weight function $w : E \rightarrow \mathbb{R}_+$, k distinct terminal vertices $T := \{t_1, \dots, t_k\} \subseteq V$, and a value $D > 0$ such that there exists a partition (P_1^*, \dots, P_k^*) of V with $t_i \in P_i^*$ for all $i \in [k]$ such that $\sum_{i=1}^k w(\delta(P_i^*))^p \leq D$, and runs in polynomial time to return a multiway cut $\mathcal{P} = (P_1, \dots, P_k)$ such that

$$\left(\sum_{i=1}^k w(\delta(P_i))^p \right)^{\frac{1}{p}} = (O((\beta \log n)^p D))^{\frac{1}{p}} = O(\beta \log n) D^{\frac{1}{p}} = O(\log^{1.5} n \log^{0.5} k) D^{\frac{1}{p}}.$$

In order to prove Theorem 2, we may use binary search to guess $D \in [\text{OPT}^p, (2\text{OPT})^p]$ and run the above algorithm to obtain a multiway cut $\mathcal{P} = (P_1, \dots, P_k)$ such that

$$\left(\sum_{i=1}^k w(\delta(P_i))^p \right)^{\frac{1}{p}} = O(\log^{1.5} n \log^{0.5} k) D^{\frac{1}{p}} = O(\log^{1.5} n \log^{0.5} k) \text{OPT}.$$

This completes the proof of Theorem 2.

4 Convex program and integrality gap

The following is a natural convex programming relaxation for ℓ_p -NORM-MULTIWAY-CUT on instance (G, w, T) where $T = \{t_1, \dots, t_k\}$ are the terminal vertices (the objective function can be convexified by introducing additional variables and constraints):

$$\begin{aligned} & \text{Minimize} \quad \left(\sum_{i=1}^k \left(\sum_{uv \in E} w(uv) \cdot |x(u, i) - x(v, i)| \right)^p \right)^{1/p} \quad \text{subject to} \\ & \sum_{i=1}^k x(v, i) = 1 \quad \forall v \in V, \\ & x(t_i, i) = 1 \quad \forall i \in [k], \\ & x(v, i) \geq 0 \quad \forall v \in V, \forall i \in [k]. \end{aligned} \tag{1}$$

► **Lemma 12.** *The convex program in (1) has an integrality gap of at least $k^{1-1/p}/2$.*

Proof. Consider the star graph that has k leaves $\{t_1, \dots, t_k\}$ and a center vertex v with all edge weights being 1. Let the terminal vertices be the k leaves. The optimum ℓ_p -norm objective value of a multiway cut is

$$((k-1)^p + k - 1)^{\frac{1}{p}},$$

and it corresponds to the partition $(\{t_1, v\}, \{t_2\}, \{t_3\}, \dots, \{t_k\})$. A feasible solution to the convex program (1) is given by $x(v, i) = 1/k$ for all $i \in [k]$, which yields an objective of

$$\left(k \cdot \left(\frac{k-1}{k} + (k-1) \cdot \frac{1}{k} \right)^p \right)^{\frac{1}{p}} = \frac{2k-2}{k} \cdot k^{\frac{1}{p}}.$$

This results in an integrality gap of at least

$$\frac{((k-1)^p + k - 1)^{\frac{1}{p}}}{\frac{2k-2}{k} \cdot k^{\frac{1}{p}}} \geq \frac{k-1}{\frac{2k-2}{k} \cdot k^{\frac{1}{p}}} = \frac{k^{1-\frac{1}{p}}}{2}. \quad \blacktriangleleft$$

Bansal et al. give an SDP relaxation for MIN-MAX-MULTIWAY-CUT and show that the star graph has an integrality gap of $\Omega(k)$ for this SDP relaxation. This SDP relaxation can be generalized in a natural fashion to ℓ_p -NORM-MULTIWAY-CUT. The star graph still exhibits an integrality gap of $\Omega(k^{1-1/p})$ for the generalized SDP relaxation for ℓ_p -NORM-MULTIWAY-CUT.

5 Conclusion

In this work, we introduced ℓ_p -NORM-MULTIWAY-CUT for $p \geq 1$ as a unified generalization of MULTIWAY-CUT and MIN-MAX-MULTIWAY-CUT. We showed that ℓ_p -NORM-MULTIWAY-CUT is NP-hard for constant number of terminals or in planar graphs for every $p \geq 1$. The natural convex program for ℓ_p -NORM-MULTIWAY-CUT has an integrality gap of $\Omega(k^{1-1/p})$ and the problem is $(k^{1-1/p-\epsilon})$ -inapproximable for any constant $\epsilon > 0$ assuming the small set expansion hypothesis, where k is the number of terminals in the input instance. The inapproximability result suggests that a dependence on n in the approximation factor is unavoidable if we would like to obtain an approximation factor that is better than the trivial $O(k^{1-1/p})$ -factor. On the algorithmic side, we gave an $O(\sqrt{\log^3 n \log k})$ -approximation (i.e., an $O(\log^2 n)$ -approximation), where n is the number of vertices in the input graph. Our results suggest that the approximability behaviour of ℓ_p -NORM-MULTIWAY-CUT exhibits a sharp transition from $p = 1$ to $p > 1$. Our work raises several open questions. We mention a couple of them: (1) Can we achieve an $O(\log n)$ -approximation for ℓ_p -NORM-MULTIWAY-CUT for every $p \geq 1$? We recall that when $p = \infty$, the current best approximation factor is indeed $O(\log n)$ [3]. (2) Is there a polynomial-time algorithm for ℓ_p -NORM-MULTIWAY-CUT for any given p that achieves an approximation factor that smoothly interpolates between the best possible approximation for $p = 1$ and the best possible approximation for $p = \infty$ – e.g., is there an $O(\log^{1-1/p} n)$ -approximation?

References


- 1 S. Ahmadi, S. Khuller, and B. Saha. Min-max correlation clustering via multicut. In *Integer Programming and Combinatorial Optimization*, IPCO, pages 13–26, 2019.
- 2 H. Angelidakis, Y. Makarychev, and P. Manurangsi. An improved integrality gap for the Călinescu-Karloff-Rabani relaxation for multiway cut. In *Integer Programming and Combinatorial Optimization*, IPCO, pages 39–50, 2017.
- 3 N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. Naor, and R. Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.
- 4 K. Bérczi, K. Chandrasekaran, T. Király, and V. Madan. Improving the integrality gap for multiway cut. *Mathematical Programming*, 183:171–193, 2020.
- 5 N. Buchbinder, J. Naor, and R. Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. In *Proceedings of the forty-fifth annual ACM Symposium on Theory of Computing*, STOC, pages 535–544, 2013.
- 6 N. Buchbinder, R. Schwartz, and B. Weizman. Simplex transformations and the multiway cut problem. In *Proceedings of the twenty-eighth annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2400–2410, 2017.
- 7 G. Călinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60(3):564–574, 2000.
- 8 K. Chandrasekaran and C. Chekuri. Min-max partitioning of hypergraphs and symmetric submodular functions. In *Proceedings of the thirty-second annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1026–1038, 2021.
- 9 K. Chandrasekaran and W. Wang. ℓ_p -norm Multiway Cut. Preprint in arXiv: 2106.14840, 2021.
- 10 M. Charikar, N. Gupta, and R. Schwartz. Local guarantees in graph cuts and clustering. In *Integer Programming and Combinatorial Optimization*, IPCO, pages 136–147, 2017.
- 11 C. Chekuri and A. Ene. Submodular cost allocation problem and applications. In *International Colloquium on Automata, Languages and Programming*, ICALP, pages 354–366, 2011.

- 12 K. Cheung, W. Cunningham, and L. Tang. Optimal 3-terminal cuts and linear programming. *Mathematical Programming*, 106(1):1–23, March 2006.
- 13 V. Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.
- 14 E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.
- 15 S. Kalhan, K. Makarychev, and T. Zhou. Correlation clustering with local objectives. In *Advances in Neural Information Processing Systems 32*, pages 9346–9355, 2019.
- 16 D. Karger, P. Klein, C. Stein, M. Thorup, and N. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research*, 29(3):436–461, 2004.
- 17 R. Manokaran, J. Naor, P. Raghavendra, and R. Schwartz. SDP gaps and UGC hardness for multiway cut, 0-extension, and metric labeling. In *Proceedings of the fortieth annual ACM Symposium on Theory of Computing*, STOC, pages 11–20, 2008.
- 18 D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.
- 19 G. Puleo and O. Milenkovic. Correlation clustering and biclustering with locally bounded errors. *IEEE Transactions on Information Theory*, 64:4105–4119, 2018.
- 20 P. Raghavendra, D. Steurer, and M. Tulsiani. Reductions between expansion problems. In *IEEE Conference on Computational Complexity*, CCC, pages 64–73, 2012.
- 21 A. Sharma and J. Vondrák. Multiway cut, pairwise realizable distributions, and descending thresholds. In *Proceedings of the forty-sixth annual ACM Symposium on Theory of Computing*, STOC, pages 724–733, 2014.
- 22 Z. Svitkina and É. Tardos. Min-max multiway cut. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX, pages 207–218, 2004.

Faster Algorithms for Longest Common Substring

Panagiotis Charalampopoulos ✉ 

The Interdisciplinary Center Herzliya, Israel


Tomasz Kociumaka ✉ 

University of California, Berkeley, CA, USA

Solon P. Pissis ✉ 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski ✉ 

Institute of Informatics, University of Warsaw, Poland

Samsung R&D, Warsaw, Poland

Abstract

In the classic longest common substring (LCS) problem, we are given two strings S and T , each of length at most n , over an alphabet of size σ , and we are asked to find a longest string occurring as a fragment of both S and T . Weiner, in his seminal paper that introduced the suffix tree, presented an $\mathcal{O}(n \log \sigma)$ -time algorithm for this problem [SWAT 1973]. For polynomially-bounded integer alphabets, the linear-time construction of suffix trees by Farach yielded an $\mathcal{O}(n)$ -time algorithm for the LCS problem [FOCS 1997]. However, for small alphabets, this is not necessarily optimal for the LCS problem in the word RAM model of computation, in which the strings can be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time. We show that, in this model, we can compute an LCS in time $\mathcal{O}(n \log \sigma / \sqrt{\log n})$, which is sublinear in n if $\sigma = 2^{o(\sqrt{\log n})}$ (in particular, if $\sigma = \mathcal{O}(1)$), using optimal space $\mathcal{O}(n \log \sigma / \log n)$.

We then lift our ideas to the problem of computing a k -mismatch LCS, which has received considerable attention in recent years. In this problem, the aim is to compute a longest substring of S that occurs in T with at most k mismatches. Flouri et al. showed how to compute a 1-mismatch LCS in $\mathcal{O}(n \log n)$ time [IPL 2015]. Thankachan et al. extended this result to computing a k -mismatch LCS in $\mathcal{O}(n \log^k n)$ time for $k = \mathcal{O}(1)$ [J. Comput. Biol. 2016]. We show an $\mathcal{O}(n \log^{k-1/2} n)$ -time algorithm, for any constant integer $k > 0$ and *irrespective* of the alphabet size, using $\mathcal{O}(n)$ space as the previous approaches. We thus notably break through the well-known $n \log^k n$ barrier, which stems from a recursive heavy-path decomposition technique that was first introduced in the seminal paper of Cole et al. [STOC 2004] for string indexing with k errors.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases longest common substring, k mismatches, wavelet tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.30

Related Version *Full Version:* <https://arxiv.org/abs/2105.03106>

Funding *Panagiotis Charalampopoulos:* Supported by the Israel Science Foundation grant 592/17. *Tomasz Kociumaka:* Partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

Solon P. Pissis: This paper is part of the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539. This paper is also part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.



© Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 30; pp. 30:1–30:17

Leibniz International Proceedings in Informatics



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

1 Introduction

In the classic longest common substring (LCS) problem, we are given two strings S and T , each of length at most n , over an alphabet of size σ , and we are asked to find a longest string occurring as a fragment of both S and T . The problem was conjectured by Knuth to require $\Omega(n \log n)$ time until Weiner, in his seminal paper introducing the suffix tree [62], showed that the LCS problem can be solved in $\mathcal{O}(n)$ time when σ is constant via constructing the suffix tree of string $S\#T$, for a sentinel letter $\#$. Later, Farach showed that if σ is not constant, the suffix tree can be constructed in linear time in addition to the time required for sorting its letters [33]. This yielded an $\mathcal{O}(n)$ -time algorithm for the LCS problem in the word RAM model for polynomially-bounded integer alphabets. While Farach's algorithm for suffix tree construction is *optimal* for all alphabets (the suffix tree by definition has size $\Theta(n)$), the same does not hold for the LCS problem. We were thus motivated to answer the following basic question:

Can the LCS problem be solved in $o(n)$ time when $\log \sigma = o(\log n)$?

We consider the word RAM model and assume an alphabet $[0, \sigma)$. Any string of length n can then be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time. Note that if $\log \sigma = \Theta(\log n)$, one requires $\Theta(n)$ time to read the input. We answer this basic question positively when $\log \sigma = o(\sqrt{\log n})$:

► **Theorem 1.** *Given two strings S and T , each of length at most n , over an alphabet $[0, \sigma)$, we can solve the LCS problem in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n / \log_\sigma n)$ space.*

We also consider the following generalisation of the LCS problem that allows mismatches.

k -MISMATCH LONGEST COMMON SUBSTRING (k -LCS)

Input: Two strings S and T , each of length at most n , over an integer alphabet and an integer $k > 0$.

Output: A pair S', T' of substrings of S and T , respectively, with Hamming distance (i.e., number of mismatches) at most k and maximal length.

Flouri et al. presented an $\mathcal{O}(n \log n)$ -time algorithm for the 1-LCS problem [35]. (Earlier work on this problem includes [6].) This was generalised by Thankachan et al. [59] to an algorithm for the k -LCS problem that works in $\mathcal{O}(n \log^k n)$ time if $k = \mathcal{O}(1)$. Both algorithms use $\mathcal{O}(n)$ space. In [24], Charalampopoulos et al. presented an $\mathcal{O}(n + n \log^{k+1} n / \sqrt{\ell})$ -time algorithm for k -LCS with $k = \mathcal{O}(1)$, where ℓ is the length of a k -LCS. For general k , Flouri et al. presented an $\mathcal{O}(n^2)$ -time algorithm that uses $\mathcal{O}(1)$ additional space [35]. Grabowski [40] presented two algorithms with running times $\mathcal{O}(n((k+1)(\ell_0+1))^k)$ and $\mathcal{O}(n^2 k / \ell_k)$, where ℓ_0 and ℓ_k are, respectively, the length of an LCS of S and T and the length of a k -LCS of S and T . Abboud et al. [1] employed the polynomial method to obtain a $k^{1.5} n^2 / 2^{\Omega(\sqrt{\log n/k})}$ -time randomised algorithm. In [49], Kociumaka et al. showed that, assuming the Strong Exponential Time Hypothesis (SETH) [45, 46], no strongly subquadratic-time solution for k -LCS exists for $k = \Omega(\log n)$. The authors of [49] additionally presented a subquadratic-time 2-approximation algorithm for k -LCS for general k .

Analogously to Weiner's solution to the LCS problem via suffix trees, the algorithm of Thankachan et al. [59] builds upon the ideas of the k -errata tree, which was introduced by Cole et al. [29] in their seminal paper for indexing a string of length n with the aim of answering pattern matching queries with up to k mismatches. For constant k , the size of the k -errata tree is $\mathcal{O}(n \log^k n)$. (Note that computing a k -LCS using the k -errata tree directly is not straightforward as opposed to computing an LCS using the suffix tree.)

We show the following result, breaking through the $n \log^k n$ barrier, for any constant integer $k > 0$ and irrespective of the alphabet size. Recall that, in the word RAM, the letters of S and T can be renumbered in $\mathcal{O}(n \log \log n)$ time [42] so that they belong to $[0, \sigma)$.

► **Theorem 2.** *Given two strings S and T , each of length at most n , over an integer alphabet and a constant integer $k > 0$, the k -LCS problem can be solved in $\mathcal{O}(n \log^{k-1/2} n)$ time using $\mathcal{O}(n)$ space.*

Notably, on the way to proving the above theorem, we improve upon [24] by showing an $\mathcal{O}(n + n \log^{k+1} n / \ell)$ -time algorithm for k -LCS with $k = \mathcal{O}(1)$, where ℓ is the length of a k -LCS. (Our second summand is smaller by a $\sqrt{\ell}$ multiplicative factor compared to [24].)

Our Techniques

At the heart of our approaches lies the following TWO STRING FAMILIES LCP PROBLEM. (Here, the length of the longest common prefix of two strings U and V is denoted by $\text{LCP}(U, V)$; see Preliminaries for a precise definition of compacted tries.)

TWO STRING FAMILIES LCP PROBLEM

Input: Compacted tries $\mathcal{T}(\mathcal{F}_1), \mathcal{T}(\mathcal{F}_2)$ of $\mathcal{F}_1, \mathcal{F}_2 \subseteq \Sigma^*$ and two sets $\mathcal{P}, \mathcal{Q} \subseteq \mathcal{F}_1 \times \mathcal{F}_2$, with $|\mathcal{P}|, |\mathcal{Q}|, |\mathcal{F}_1|, |\mathcal{F}_2| \leq N$.

Output: The value

$$\text{maxPairLCP}(\mathcal{P}, \mathcal{Q}) = \max\{\text{LCP}(P_1, Q_1) + \text{LCP}(P_2, Q_2) : (P_1, P_2) \in \mathcal{P}, (Q_1, Q_2) \in \mathcal{Q}\}.$$

This abstract problem was introduced in [24]. Its solution, shown in the lemma below, is directly based on a technique that was used in [19, 31] and then in [35] to devise an $\mathcal{O}(n \log n)$ -time solution for 1-LCS. In particular, Lemma 3 immediately implies an $\mathcal{O}(n \log n)$ -time algorithm for 1-LCS.

► **Lemma 3** ([24, Lemma 3]). *The TWO STRING FAMILIES LCP PROBLEM can be solved in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space.¹*

In the algorithm underlying Lemma 3, for each node v of $\mathcal{T}(\mathcal{F}_1)$ we try to identify a pair of elements, one from \mathcal{P} and one from \mathcal{Q} , whose first components are descendants of v and the LCP of their second components is maximised. The algorithm traverses $\mathcal{T}(\mathcal{F}_1)$ bottom-up and uses mergeable height-balanced trees with $\mathcal{O}(N \log N)$ total merging time to store elements of pairs; see [20].

An $o(N \log N)$ time solution to the TWO STRING FAMILIES LCP PROBLEM is not known and devising such an algorithm seems difficult. The key ingredient of our algorithms is an efficient solution to the following special case of the problem. We say that a family of string pairs \mathcal{P} is an (α, β) -family if each $(U, V) \in \mathcal{P}$ satisfies $|U| \leq \alpha$ and $|V| \leq \beta$.

► **Lemma 4.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which \mathcal{P} and \mathcal{Q} are (α, β) -families can be solved in time $\mathcal{O}(N(\alpha + \log N)(\log \beta + \sqrt{\log N}) / \log N)$ and space $\mathcal{O}(N + N\alpha / \log N)$.*

The algorithm behind this solution uses a wavelet tree of the first components of $\mathcal{P} \cup \mathcal{Q}$.

¹ The original formulation of [24, Lemma 3] does not include a claim about the space complexity. However, it can be readily verified that the underlying algorithm, described in [31, 35], uses only linear space.

Solution to LCS. For the LCS problem, we design three different algorithms depending on the length of the solution. For short LCS ($\leq \frac{1}{3} \log_\sigma n$), we employ a simple tabulation technique. For long LCS ($\geq \log^4 n$), we employ the technique of Charalampopoulos et al. [24] for computing a long k -LCS, plugging in the sublinear longest common extension (LCE) data structure of Kempa and Kociumaka [47]. Both of these solutions work in $\mathcal{O}(n/\log_\sigma n)$ time.

As for medium-length LCS, let us first consider a case when the strings do not contain highly periodic fragments. In this case, we use the string synchronising sets of Kempa and Kociumaka [47] to select a set of $\mathcal{O}(\frac{n}{\tau})$ anchors over S and T , where $\tau = \Theta(\log_\sigma n)$, such that for any common substring U of S and T of length $\ell \geq 3\tau - 1$, there exist occurrences $S[i^S \dots j^S]$ and $T[i^T \dots j^T]$ of U , for which we have anchors $a^S \in [i^S, j^S]$ and $a^T \in [i^T, j^T]$ with $a^S - i^S = a^T - i^T \leq \tau$. For each anchor a in S , we add a string pair $((S[a - \tau \dots a])^R, S[a \dots a + \beta])$ to \mathcal{P} (and similarly for T and \mathcal{Q}). This lets us apply Lemma 4 with $N = \mathcal{O}(n/\tau)$, $\alpha = \mathcal{O}(\tau)$, and $\beta = \mathcal{O}(\log^4 n)$. In the periodic case, we cannot guarantee that $a^S - i^S = a^T - i^T$ is small, but we can obtain a different set of anchors based on maximal repetitions (runs) that yields multiple instances of the TWO STRING FAMILIES LCP PROBLEM, which have extra structure leading to a linear-time solution.

Solution to k -LCS. In this case we also obtain a set of $\mathcal{O}(n/\ell)$ anchors, where ℓ is the length of k -LCS. If the common substring is far from highly periodic, we use a synchronising set for $\tau = \Theta(\ell)$, and otherwise we generate anchors using a technique of misperiods that was initially introduced for k -mismatch pattern matching [18, 26]. Now the families \mathcal{P}, \mathcal{Q} need to consist not simply of substrings of S and T , but rather of modified substrings generated by an approach that resembles k -errata trees [29]. This requires combining the ideas of Thankachan et al. [59] and Charalampopoulos et al. [24]; this turns out to be technically challenging in order to stay within linear space. We apply Lemma 3 or Lemma 4 depending on the length ℓ , which allows us to break through the $n \log^k n$ barrier for k -LCS.

Other Related Work

A large body of work has been devoted to exploiting bit-parallelism in the word RAM model for string matching [7, 57, 55, 36, 37, 48, 14, 21, 9, 15, 11, 17, 39, 12, 16].

Other results on the LCS problem include the linear-time computation of an LCS of several strings over an integer alphabet [43], trade-offs between the time and the working space for computing an LCS of two strings [13, 50, 56], and the dynamic maintenance of an LCS [2, 3, 25]. Very recently, a strongly sublinear-time quantum algorithm and a lower bound for the quantum setting were shown [38]. The k -LCS problem has also been studied under edit distance and subquadratic-time algorithms for $k = o(\log n)$ are known [58, 4].

The problem of indexing a string of length n over an alphabet $[0, \sigma)$ in sublinear time in the word RAM model, with the aim of answering pattern matching queries, has attracted significant attention. Since by definition the suffix tree occupies $\Theta(n)$ space, alternative indexes have been sought. The state of the art is an index that occupies $\mathcal{O}(n \log \sigma / \log n)$ space and can be constructed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time [47, 53]. Interestingly, the running time of our algorithm (Theorem 1) matches the construction time of this index. Note that, in contrast to suffix trees, such indexes *cannot be used directly* for computing an LCS. Intuitively, these indexes sample suffixes of the string to be indexed, and upon a pattern matching query, they have to treat separately the first $\mathcal{O}(\log_\sigma n)$ letters of the pattern.

As for k -mismatch indexing, for $k = \mathcal{O}(1)$, a k -errata tree occupies $\mathcal{O}(n \log^k n)$ space, can be constructed in $\mathcal{O}(n \log^{k+1} n)$ time, and supports pattern matching queries with at most k mismatches in $\mathcal{O}(m + \log^k n \log \log n + occ)$ time, where m is the length of the pattern and occ

is the number of the reported pattern occurrences. Other trade-offs for this problem, in which the product of space and query time is still $\Omega(n \log^{2k} n)$, were shown in [23, 60], and solutions with $\mathcal{O}(n)$ space but $\Omega(\min\{n, \sigma^k m^{k-1}\})$ -time queries were presented in [22, 27, 44, 61]. More efficient solutions for $k = 1$ are known (see [10] and references therein). Cohen-Addad et al. [28] showed that, under SETH, for $k = \Theta(\log n)$ any indexing data structure that can be constructed in polynomial time cannot have $\mathcal{O}(n^{1-\delta})$ query time, for any $\delta > 0$. They also showed that in the pointer machine model, for $k = o(\log n)$, exponential dependency on k either in the space or in the query time cannot be avoided (for the reporting version of the problem). We hope that our techniques can fuel further progress in k -mismatch indexing.

2 Preliminaries

Strings. Let $T = T[1]T[2] \cdots T[n]$ be a *string* (or *text*) of length $n = |T|$ over an alphabet $\Sigma = [0, \sigma)$. The elements of Σ are called *letters*.

By ε we denote the *empty string*. For two positions i and j of T , we denote by $T[i..j]$ the *fragment* of T that starts at position i and ends at position j (the fragment is empty if $i > j$). A fragment of T is represented using $\mathcal{O}(1)$ space by specifying the indices i and j . We define $T[i..j) = T[i..j-1]$ and $T(i..j] = T[i+1..j]$. The fragment $T[i..j]$ is an *occurrence* of the underlying *substring* $P = T[i] \cdots T[j]$. We say that P occurs at *position* i in T . A *prefix* of T is a fragment of T of the form $T[1..j]$ and a *suffix* of T is a fragment of T of the form $T[i..n]$. We denote the *reverse string* of T by T^R , i.e., $T^R = T[n]T[n-1] \cdots T[1]$. By UV we denote the *concatenation* of two strings U and V , i.e., $UV = U[1]U[2] \cdots U[|U|]V[1]V[2] \cdots V[|V|]$.

A positive integer p is called a *period* of a string T if $T[i] = T[i+p]$ for all $i \in [1, |T| - p]$. We refer to the smallest period as *the period* of the string, and denote it by $\text{per}(T)$. A string T is called *periodic* if $\text{per}(T) \leq |T|/2$ and *aperiodic* otherwise. A *run* in T is a periodic substring that cannot be extended (to the left nor to the right) without an increase of its shortest period. All runs in a string can be computed in linear time [8, 51, 32].

► **Lemma 5** (Periodicity Lemma (weak version) [34]). *If a string S has periods p and q such that $p + q \leq |S|$, then $\gcd(p, q)$ is also a period of S .*

Tries. Let \mathcal{M} be a finite set containing $m > 0$ strings over Σ . The *trie* of \mathcal{M} , denoted by $\mathcal{R}(\mathcal{M})$, contains a node for every distinct prefix of a string in \mathcal{M} ; the root node is ε ; the set of leaf nodes is \mathcal{M} ; and edges are of the form $(u, \alpha, u\alpha)$, where u and $u\alpha$ are nodes and $\alpha \in \Sigma$. The *compacted trie* of \mathcal{M} , denoted by $\mathcal{T}(\mathcal{M})$, contains the root, the branching nodes, and the leaf nodes of $\mathcal{R}(\mathcal{M})$. Each maximal branchless path segment from $\mathcal{R}(\mathcal{M})$ is replaced by a single edge, and a fragment of a string $M \in \mathcal{M}$ is used to represent the label of this edge in $\mathcal{O}(1)$ space. The best known example of a compacted trie is the suffix tree [62]. Throughout our algorithms, \mathcal{M} always consists of a set of substrings or modified substrings with $k = \mathcal{O}(1)$ modifications (see Section 5 for a definition) of a reference string. The value $\text{val}(u)$ of a node u is the concatenation of labels of edges on the path from the root to u , and the *string-depth* of u is the length of $\text{val}(u)$. The size of $\mathcal{T}(\mathcal{M})$ is $\mathcal{O}(m)$. We use the following well-known construction (cf. [30]).

► **Lemma 6.** *Given a sorted list of N strings and the longest common prefixes between pairs of consecutive strings, the compacted trie of the strings can be constructed in $\mathcal{O}(N)$ time.*

Packed strings. We assume the unit-cost word RAM model with word size $w = \Theta(\log n)$ and a standard instruction set including arithmetic operations, bitwise Boolean operations, and shifts. We count the space complexity of our algorithms in machine words used by the

algorithm. The *packed representation* of a string T over alphabet $[0, \sigma)$ is a list obtained by storing $\Theta(\log_\sigma n)$ letters per machine word thus representing T in $\mathcal{O}(|T|/\log_\sigma n)$ machine words. If T is given in the packed representation we simply say that T is a *packed string*.

String synchronising sets. Our solution uses the string synchronising sets introduced by Kempa and Kociumaka [47]. Informally, in the simpler case that T is cube-free, a τ -synchronising set of T is a small set of *synchronising positions* in T such that each length- τ fragment of T contains at least one synchronising position, and the leftmost synchronising positions within two sufficiently long matching fragments of T are consistent.

Formally, for a string T and a positive integer $\tau \leq \frac{1}{2}n$, a set $A \subseteq [1, n - 2\tau + 1]$ is a τ -synchronising set of T if it satisfies the following two conditions:

1. If $T[i..i + 2\tau] = T[j..j + 2\tau]$, then $i \in A$ if and only if $j \in A$.
2. For $i \in [1, n - 3\tau + 2]$, $A \cap [i, i + \tau] = \emptyset$ if and only if $\text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau$.

► **Theorem 7** ([47, Proposition 8.10, Theorem 8.11]). *For a string $T \in [0, \sigma)^n$ with $\sigma = n^{\mathcal{O}(1)}$ and $\tau \leq \frac{1}{2}n$, there exists a τ -synchronising set of size $\mathcal{O}(n/\tau)$ that can be constructed in $\mathcal{O}(n)$ time or, if $\tau \leq \frac{1}{5}\log_\sigma n$, in $\mathcal{O}(n/\tau)$ time if T is given in a packed representation.*

As in [47], for a τ -synchronising set A , let $\text{succ}_A(i) := \min\{j \in A \cup \{n - 2\tau + 2\} : j \geq i\}$.

► **Lemma 8** ([47, Fact 3.2]). *If $p = \text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau$, then $T[i.. \text{succ}_A(i) + 2\tau - 1]$ is the longest prefix of $T[i..|T|]$ with period p .*

► **Lemma 9** ([47, Fact 3.3]). *If a string U with $|U| \geq 3\tau - 1$ and $\text{per}(U) > \frac{1}{3}\tau$ occurs at positions i and j in T , then $\text{succ}_A(i) - i = \text{succ}_A(j) - j \leq |U| - 2\tau$.*

A τ -run R is a run of length at least $3\tau - 1$ with period at most $\frac{1}{3}\tau$. The *Lyndon root* of R is the lexicographically smallest cyclic shift of $R[1.. \text{per}(R)]$. A proof of the following lemma resembles an argument given in [47, Section 6.1.2]; its proof can be found in the full version of this paper.

► **Lemma 10.** *For a positive integer τ , a string $T \in [0, \sigma)^n$ contains $\mathcal{O}(n/\tau)$ τ -runs. Moreover, if $\tau \leq \frac{1}{5}\log_\sigma n$, given a packed representation of T , we can compute all τ -runs in T and group them by their Lyndon roots in $\mathcal{O}(n/\tau)$ time. Within the same complexities, for each τ -run, we can compute the two leftmost occurrences of its Lyndon root.*

► **Theorem 11** ([47, Theorem 4.3]). *Given a packed representation of a string $T \in [0, \sigma)^n$ and a τ -synchronising set A of T of size $\mathcal{O}(n/\tau)$ for $\tau = \mathcal{O}(\log_\sigma n)$, we can compute in $\mathcal{O}(n/\tau)$ time the lexicographic order of all suffixes of T starting at positions in A .*

We often want to preprocess T to be able to answer queries $\text{LCP}(T[i..n], T[j..n])$ [52]. For this case, there exists an optimal data structure that applies synchronising sets.

► **Theorem 12** ([47, Theorem 5.4]). *Given a packed representation of a string $T \in [0, \sigma)^n$, LCP queries on T can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$ -time preprocessing.*

3 Sublinear-Time LCS

We provide different solutions depending on the length ℓ of an LCS. Lemmas 13, 14, and 17 directly yield Theorem 1.

The proof of the following lemma, for the case where an LCS is short, i.e., of length $\ell \leq \frac{1}{3}\log_\sigma n$, uses tabulation and can be found in the full version of this paper.

► **Lemma 13.** *The LCS problem can be solved in $\mathcal{O}(n/\log_\sigma n)$ time if $\ell \leq \frac{1}{3} \log_\sigma n$.*

The proof of the following lemma, for the case where an LCS is long, i.e., of length $\ell = \Omega(\frac{\log^4 n}{\log^2 \sigma})$, uses difference covers and the $\mathcal{O}(N \log N)$ -time solution to the TWO STRING FAMILIES LCP PROBLEM. This proof closely follows [24] and can be found in the full version of this paper.

► **Lemma 14.** *The LCS problem can be solved in $\mathcal{O}(n/\log_\sigma n)$ time if $\ell = \Omega(\frac{\log^4 n}{\log^2 \sigma})$.*

Solution for medium-length LCS. We now give a solution to the LCS problem for $\ell \in [\frac{1}{3} \log_\sigma n, 2\sqrt{\log n}]$. We first construct three subsets of positions in $S\$T$, where $\$ \notin \Sigma$, of size $\mathcal{O}(n/\log_\sigma n)$ as follows. For $\tau = \lfloor \frac{1}{9} \log_\sigma n \rfloor$, let A_I be a τ -synchronising set of $S\$T$. For each τ -run in $S\$T$, we insert to A_{II} the starting positions of the first two occurrences of the Lyndon root of the τ -run and to A_{III} the last position of the τ -run. The elements of A_{II} and A_{III} store the τ -run they originate from. Finally, we denote $A_j^S = A_j \cap [1, |S|]$ and $A_j^T = \{a - |S| - 1 : a \in A_j, a > |S| + 1\}$ for $j = I, II, III$. The following lemma shows that there exists an LCS of S and T for which $A_I \cup A_{II} \cup A_{III}$ is a set of *anchors* that satisfies certain distance requirements.

► **Lemma 15.** *If an LCS of S and T has length $\ell \geq 3\tau$, then there exist positions $i^S \in [1, |S|]$, $i^T \in [1, |T|]$, a shift $\delta \in [0, \ell)$, and $j \in \{I, II, III\}$ such that $S[i^S \dots i^S + \ell) = T[i^T \dots i^T + \ell)$, $i^S + \delta \in A_j^S$, $i^T + \delta \in A_j^T$, and*

- *if $j = I$, then $\delta \in [0, \tau)$;*
- *if $j = II$, then $S[i^S \dots i^S + \ell)$ is contained in the τ -run from which $i^S + \delta \in A^S$ originates;*
- *if $j = III$, then $\delta \geq 3\tau - 1$ and $S[i^S \dots i^S + \delta]$ is a suffix of the τ -run from which $i^S + \delta \in A^S$ originates.*

Proof. By the assumption, there exist $i^S \in [1, |S|]$ and $i^T \in [1, |T|]$ such that $S[i^S \dots i^S + \ell) = T[i^T \dots i^T + \ell)$. Let us choose any such pair (i^S, i^T) minimising the sum $i^S + i^T$. We have the following cases.

1. If $\text{per}(S[i^S \dots i^S + 3\tau - 2]) > \frac{1}{3}\tau$, then, by the definition of a τ -synchronising set, in this case there exist some elements $a^S \in A_I^S \cap [i^S, i^S + \tau)$ and $a^T \in A_I^T \cap [i^T, i^T + \tau)$. Let us choose the smallest such elements. By Lemma 9, we have $a^S - i^S = a^T - i^T$.
2. Else, $p = \text{per}(S[i^S \dots i^S + 3\tau - 2]) \leq \frac{1}{3}\tau$. We have two subcases.
 - a. If $p = \text{per}(S[i^S \dots i^S + \ell))$, then, by the choice of i^S and i^T there exists a τ -run R_S in S that starts at position in $(i^S - p \dots i^S]$ and a τ -run R_T in T that starts at position in $(i^T - p \dots i^T]$. Moreover, by Lemma 5, both runs have equal Lyndon roots. For each $X \in \{S, T\}$, let us choose a^X as the leftmost starting position of a Lyndon root of R_X that is $\geq i^X$. We have $a^S - i^S = a^T - i^T \in [0, \frac{1}{3}\tau)$. Each position a^X will be the starting position of the first or the second occurrence of the Lyndon root of R_S , so $a^S \in A_{II}^S$ and $a^T \in A_{II}^T$.
 - b. Else, $p \neq \text{per}(S[i^S \dots i^S + \ell))$. We have $d = \min\{b \geq p : S[i^S + b] \neq S[i^S + b - p]\} < \ell$ (and $d \geq 3\tau - 1$). In this case, $a^S = i^S + d - 1$ and $a^T = i^T + d - 1$ are the ending positions of τ -runs with period p in S and T , respectively, so $a^S \in A_{III}^S$ and $a^T \in A_{III}^T$. ◀

Case $j = I$ from the above lemma corresponds to the TWO STRING FAMILIES LCP PROBLEM with \mathcal{P} and \mathcal{Q} being $(\tau, 2\sqrt{\log n})$ -families. Let us introduce a variant of the TWO STRING FAMILIES LCP PROBLEM that intuitively corresponds to the case $j \in \{II, III\}$. A family of string pairs \mathcal{P} is called a *prefix family* if there exists a string Y such that, for each

$(U, V) \in \mathcal{P}$, U is a prefix of Y . We arrive at this special case with first components of \mathcal{P} and \mathcal{Q} being prefixes of some cyclic shift of a power of a (common) Lyndon root of τ -runs. The proof of the following lemma can be found in the full version of this paper.

► **Lemma 16.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which $\mathcal{P} \cup \mathcal{Q}$ is a prefix family can be solved in $\mathcal{O}(N)$ time.*

We are now ready to state the main result of this subsection.

► **Lemma 17.** *The LCS problem can be solved in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n / \log_\sigma n)$ space if $\ell \in [\frac{1}{3} \log_\sigma n, 2^{\sqrt{\log n}}]$.*

Proof. Recall that $\tau = \lfloor \frac{1}{9} \log_\sigma n \rfloor$. The set of anchors $A = A_I \cup A_{II} \cup A_{III}$ consists of a τ -synchronising set and of $\mathcal{O}(1)$ positions per each τ -run in $S\$T$. Hence, $|A| = \mathcal{O}(n/\tau)$ and A can be constructed in $\mathcal{O}(n/\tau)$ time by Theorem 7 and Lemma 10.

We construct sets of pairs of substrings of $X = S\$_1T\$_2S^R\$_3T^R$. First, for $\Delta = \lfloor 2^{\sqrt{\log n}} \rfloor$:

$$\mathcal{P}_I = \{((S[a - \tau \dots a))^R, S[a \dots a + \Delta]) : a \in A_I^S\}.$$

Then, for each group \mathcal{G} of τ -runs in S and T with equal Lyndon root, we construct the following set of string pairs:

$$\mathcal{P}_{II}^{\mathcal{G}} = \{((S[x \dots a))^R, S[a \dots y]) : a \in A_{II}^S \text{ that originates from } \tau\text{-run } S[x \dots y] \in \mathcal{G}\}.$$

We define the *tail* of a τ -run $S[i \dots j]$ with period p and Lyndon root $S[i' \dots i' + p]$ as $(j + 1 - i') \bmod p$ (and same for τ -runs in T). For each group of τ -runs in S and T with equal Lyndon roots, we group the τ -runs belonging to it by their tails. This can be done in $\mathcal{O}(n/\tau)$ time using tabulation, since the tail values are up to $\frac{1}{3}\tau$. For each group \mathcal{G} of τ -runs in S and T with equal Lyndon root and tail, we construct the following set of string pairs:

$$\mathcal{P}_{III}^{\mathcal{G}} = \{((S[x \dots y))^R, S[y \dots |S|]) : S[x \dots y] \in \mathcal{G}\}.$$

Simultaneously, we create sets \mathcal{Q}_I , $\mathcal{Q}_{II}^{\mathcal{G}}$ and $\mathcal{Q}_{III}^{\mathcal{G}}$ defined with T instead of S .

Now, it suffices to output the maximum of $\text{maxPairLCP}(\mathcal{P}_I, \mathcal{Q}_I)$, $\text{maxPairLCP}(\mathcal{P}_{II}^{\mathcal{G}}, \mathcal{Q}_{II}^{\mathcal{G}})$, and $\text{maxPairLCP}(\mathcal{P}_{III}^{\mathcal{G}}, \mathcal{Q}_{III}^{\mathcal{G}})$, where \mathcal{G} ranges over groups of τ -runs. Computing any individual maxPairLCP value can be expressed as an instance of the TWO STRING FAMILIES LCP PROBLEM provided that all the first and second components of families are represented as nodes of compacted tries. We will use Lemma 6 to construct these compacted tries. LCP queries can be answered efficiently due to Theorem 12, so it suffices to be able to sort all the first and second components of each pair of string pair sets lexicographically. Each of the sets \mathcal{P}_I , \mathcal{Q}_I can be ordered by the second components using Theorem 11 since A_I is a τ -synchronising set, and by the first components with easy preprocessing using the fact that the number of possible τ -length strings is $\sigma^\tau = \mathcal{O}(n^{1/9})$. In a set $\mathcal{P}_{II}^{\mathcal{G}}$, both all first and all second components are prefixes of a single string (a power of the common Lyndon root). Hence, they can be sorted simply by comparing their lengths. This sorting is performed simultaneously for all the families $\mathcal{P}_{II}^{\mathcal{G}}$, $\mathcal{Q}_{II}^{\mathcal{G}}$ in $\mathcal{O}(n/\tau)$ time via radix sort. Finally, to sort the second components of the sets $\mathcal{P}_{III}^{\mathcal{G}}$, $\mathcal{Q}_{III}^{\mathcal{G}}$, instead of comparing strings of the form $S[y \dots |S|]$ (and same for T), we can equivalently compare strings $S[y - 2\tau + 1 \dots |S|]$ which are known to start at positions from a τ -synchronising set by Lemma 8. This sorting is done across all groups using radix sort and Theorem 11. The correctness follows by Lemma 15.

Finally, we observe that $(\mathcal{P}_I, \mathcal{Q}_I)$ is a (τ, Δ) -family of size $N = \mathcal{O}(n/\tau)$, and thus $\text{maxPairLCP}(\mathcal{P}_I, \mathcal{Q}_I)$ can be computed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time and $\mathcal{O}(n / \log_\sigma n)$ space using Lemma 4. On the other hand, $(\mathcal{P}_{II}^g, \mathcal{Q}_{II}^g)$ and $(\mathcal{P}_{III}^g, \mathcal{Q}_{III}^g)$ are prefix families of total size $\mathcal{O}(n/\tau)$, so the corresponding instances of the TWO STRING FAMILIES LCP PROBLEM can be solved in $\mathcal{O}(n / \log_\sigma n)$ total time using Lemma 16. \blacktriangleleft

4 Proof of Lemma 4 via Wavelet Trees

Wavelet trees. For an arbitrary alphabet Σ , the *skeleton tree* for Σ is a full binary tree \mathcal{T} together with a bijection between Σ and the leaves of \mathcal{T} . For a node $v \in \mathcal{T}$, let Σ_v denote the subset of Σ that corresponds to the leaves in the subtree of v .

The \mathcal{T} -shaped *wavelet tree* of a string $T \in \Sigma^*$ consists of bit vectors assigned to internal nodes of \mathcal{T} (inspect Figure 1(a)). For an internal node v with children v_L and v_R , let T_v denote the maximal subsequence of T that consists of letters from Σ_v ; the bit vector $B_v[1..|T_v|]$ is defined so that $B_v[i] = 0$ if $T_v[i] \in \Sigma_{v_L}$ and $B_v[i] = 1$ if $T_v[i] \in \Sigma_{v_R}$.

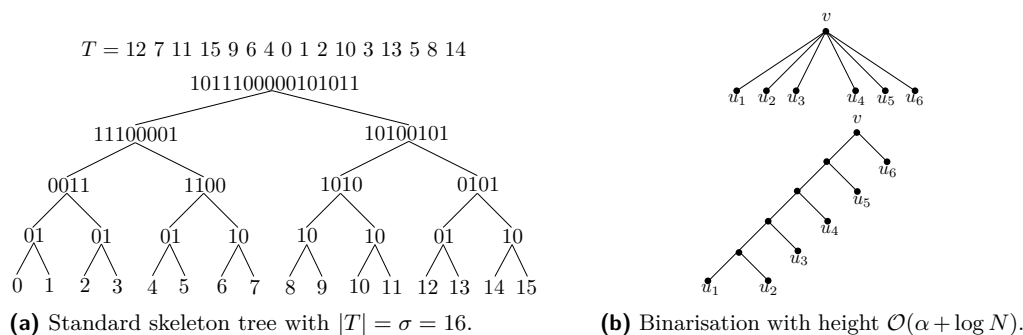


Figure 1 (a) Let v be left child of the root node. Then $\Sigma_v = \{0, 1, \dots, 7\}$, $T_v = 7, 6, 4, 0, 1, 2, 3, 5$ and so $B_v = 11100001$: 7, 6, 4, 5 belong to the right subtree of v and 0, 1, 2, 3 to the left. (b) For each i , let the size of the subtree rooted at u_i be 2^i . The binarisation from [5] leads to height $\mathcal{O}(\alpha + \log N)$, favouring heavier children.

Wavelet trees were introduced in [41], whereas efficient construction algorithms were presented in [54, 5].

► **Theorem 18** (see [54, Theorem 2]). *Given the packed representation of a string $T \in [0, \sigma]^n$ and a skeleton tree \mathcal{T} of height h , the \mathcal{T} -shaped wavelet tree of T takes $\mathcal{O}(nh / \log n + \sigma)$ space and can be constructed in $\mathcal{O}(nh / \sqrt{\log n} + \sigma)$ time.*

Wavelet trees are sometimes constructed for sequences $T \in \mathcal{M}^*$ over an alphabet $\mathcal{M} \subseteq \Sigma^*$ that itself consists of strings (see e.g. [47]). In this case, the skeleton tree \mathcal{T} is often chosen to resemble the compacted trie of \mathcal{M} . Formally, we say that a skeleton tree \mathcal{T} for \mathcal{M} is *prefix-consistent* if each node $v \in \mathcal{T}$ admits a label $\text{val}(v) \in \Sigma^*$ such that:

- if v is a leaf, then $\text{val}(v)$ is the corresponding string in \mathcal{M} ;
- if v is a node with children v_L, v_R , then, for all leaves u_L, u_R in the subtrees of v_L and v_R , respectively, the string $\text{val}(v)$ is the longest common prefix of $\text{val}(u_L)$ and $\text{val}(u_R)$.

Observe that if $\mathcal{M} \subseteq \{0, 1\}^\alpha$ for some integer α , then the compacted trie $\mathcal{T}(\mathcal{M})$ is a prefix-consistent skeleton tree for \mathcal{M} . For larger alphabets, we binarise $\mathcal{T}(\mathcal{M})$ as follows:

► **Lemma 19.** *Given the compacted trie $\mathcal{T}(\mathcal{M})$ of a set $\mathcal{M} \subseteq \Sigma^\alpha$, a prefix-consistent skeleton tree of height $\mathcal{O}(\alpha + \log |\mathcal{M}|)$ can be constructed in $\mathcal{O}(|\mathcal{M}|)$ time, with each node v associated to a node v' of $\mathcal{T}(\mathcal{M})$ such that $\text{val}(v) = \text{val}(v')$.*

Proof. We use [5, Corollary 3.2], where the authors showed that any rooted tree of size m and height h can be binarised in $\mathcal{O}(m)$ time so that the resulting tree is of height $\mathcal{O}(h + \log m)$; see Figure 1(b). For $\mathcal{T}(\mathcal{M})$, we obtain height $\mathcal{O}(\alpha + \log |\mathcal{M}|)$ and time $\mathcal{O}(|\mathcal{M}|)$. \blacktriangleleft

► **Lemma 4.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which \mathcal{P} and \mathcal{Q} are (α, β) -families can be solved in time $\mathcal{O}(N(\alpha + \log N)(\log \beta + \sqrt{\log N})/\log N)$ and space $\mathcal{O}(N + N\alpha/\log N)$.*

Proof. By traversing $\mathcal{T}(\mathcal{F}_2)$ we can compute in $\mathcal{O}(N)$ time a list \mathcal{R} being a union of sets \mathcal{P} and \mathcal{Q} in which the second components are ordered lexicographically. We also store a bit vector G of length $|\mathcal{R}|$ that determines, for each element of \mathcal{R} , which of the sets \mathcal{P} , \mathcal{Q} it originates from. We construct the wavelet tree of the sequence of strings being the first components of pairs from \mathcal{R} using Theorem 18 and the skeleton tree from Lemma 19. Before the wavelet tree is constructed, we pad each string with a letter $\$ \notin \Sigma$ to make them all of length α ; we will ignore the nodes of the wavelet tree with a path-label containing a $\$$.

For a sublist $\mathcal{X} = (U_1, V_1), \dots, (U_m, V_m)$ of \mathcal{R} , by $\text{LCPs}(\mathcal{X})$ we denote the representation of the list $0, \text{LCP}(V_1, V_2), \dots, \text{LCP}(V_{m-1}, V_m)$ as a packed string over alphabet $[0, \beta]$ in space $\mathcal{O}(N/\log_\beta N)$. Together with each $\text{LCPs}(\mathcal{X})$ we also store the bit vector $G(\mathcal{X})$ of origins of elements of \mathcal{X} without increasing the complexity. The list $\text{LCPs}(\mathcal{R})$ can be computed in $\mathcal{O}(N)$ time when constructing \mathcal{R} . For each node v of the wavelet tree, we wish to compute $\mathcal{L}_v = \text{LCPs}(\mathcal{R}_v)$, where \mathcal{R}_v is the sublist of \mathcal{R} composed of elements whose first component is in the leaf list Σ_v of v . We will construct the lists $\text{LCPs}(\mathcal{R}_v)$ without actually computing \mathcal{R}_v .

Computation of LCPs lists. The lists are computed recursively using the bit vectors from the wavelet tree. Assume we have computed \mathcal{L}_u and wish to compute \mathcal{L}_v for the left child v of u – the computations for the right child are symmetric.

Let $c \in (0, 1)$ be a constant. Let us partition \mathcal{L}_u into blocks of $\lambda = \max(1, \lfloor c \log_\beta N \rfloor)$ LCP values. We will process the blocks in order, constructing \mathcal{L}_v . Each block of \mathcal{L}_u can be represented in a single word and this representation can be extracted from the packed representation of \mathcal{L}_u in $\mathcal{O}(1)$ time. For each block $W = \mathcal{L}_u(a\lambda \dots (a+1)\lambda]$, we retrieve in $\mathcal{O}(1)$ time the corresponding block $D = B_u(a\lambda \dots (a+1)\lambda]$ in the bit vector from the wavelet tree. Further, we store $\mu_a = \min \mathcal{L}_v(p_a \dots a\lambda]$, where $p_a = \max\{i \in [0, a\lambda] : i = 0 \text{ or } B_u[i] = 0\}$. Let us show how, given W , D and μ_a , we can determine μ_{a+1} and the chunk of \mathcal{L}_v that corresponds to $i \in [1, \lambda]$ such that $D[i] = 0$. The calculations are based on the following well-known fact.

► **Fact 20.** *If U_1, U_2, U_3 are strings such that $U_1 \leq U_2 \leq U_3$, then we have $\text{LCP}(U_1, U_3) = \min(\text{LCP}(U_1, U_2), \text{LCP}(U_2, U_3))$.*

For each $i \in [1, \lambda]$ such that $D[i] = 0$, in increasing order, if a previous position j with $D[j] = 0$ exists, then $\min W(i' \dots i]$ should be appended to \mathcal{L}_v , where i' is the predecessor of i satisfying $D[i'] = 0$, and otherwise $\min(\{\mu_a\} \cup W[1 \dots i])$ should. Then, for the last position $i \in [1, \lambda]$ such that $D[i] = 0$, $\mu_{a+1} = \min W(i \dots \lambda]$, and if no such position exists, then $\mu_{a+1} = \min(\{\mu_a\} \cup W[1 \dots \lambda])$.

If $\lambda = 1$, the calculations can be performed in $\mathcal{O}(1)$ time. Otherwise, we make use of preprocessing: for every possible combination of (W, D, μ_a) , i.e., up to $2c \log N + \log \beta < 3c \log N$ bits, precompute the block to be appended to \mathcal{L}_v and μ_{a+1} , i.e., up to $c \log N + \log \beta < 2c \log N$ bits. We can choose $c > 0$ small enough to make the preprocessing $\mathcal{O}(N^{1-\varepsilon})$ for some $\varepsilon > 0$. Thus, the computation takes $\mathcal{O}(|\mathcal{L}_u|/\lambda)$ time. Within this time, we can also populate the bit vector of origins for v .

Application of LCPs lists. For each node u of the wavelet tree, we must extract the maximum LCP between suffixes of different origins, add the string-depth $|\text{val}(u)|$, and compare the result with the stored candidate. The former can be computed in $\mathcal{O}(|\mathcal{L}_u|/\log_\beta N)$ time as follows. Due to Fact 20, the answer will be the LCP between a pair of second components of consecutive elements of \mathcal{R}_u that originate from different sets, i.e. $\max\{\mathcal{L}_u[i] : i \in [2, |\mathcal{R}_u|], G(\mathcal{R}_u)[i-1] \neq G(\mathcal{R}_u)[i]\}$. We can cover all pairs of consecutive elements of \mathcal{L}_u using $\Theta(1 + |\mathcal{L}_u|/\log_\beta N)$ blocks of $\max(2, \lfloor c \log_\beta N \rfloor)$ LCP values. Each such block, augmented with its corresponding bit vector of origins, consists of at most $2c \log N$ bits. We can thus precompute all possible answers in $\mathcal{O}(N^{1-\epsilon})$ time, and then process each node u in $\Theta(1 + |\mathcal{L}_u|/\log_\beta N)$ time.

Time complexity. The wavelet tree can be built in $\mathcal{O}(Nh/\sqrt{\log N})$ time using space $\mathcal{O}(Nh/\log N)$ by Theorem 18, where $h = \mathcal{O}(\alpha + \log N)$. Computing LCPs for children of a single node u takes $\mathcal{O}(1 + |\mathcal{L}_u|/\log_\beta N)$ time; over all nodes, this is $\mathcal{O}(Nh \log \beta / \log N)$. ◀

5 Faster k -LCS

In this section, we outline our $\mathcal{O}(n \log^{k-1/2} n)$ -time algorithm for the k -LCS problem with $k = \mathcal{O}(1)$, that underlies Theorem 2. For simplicity, we focus on computing the length of a k -LCS; an actual pair of strings forming a k -LCS can be recovered easily from our approach. If the length of an LCS of S and T is d , then the length of a k -LCS of S and T is in $[d, (k+1)d + k]$. Below, we show how to compute a k -LCS provided that it belongs to an interval $(\ell/2, \ell]$ for a specified ℓ ; it is sufficient to call this subroutine $\mathcal{O}(\log k) = \mathcal{O}(1)$ times.

Similarly to our solutions for long and medium-length LCS, we first distinguish anchors $A^S \subseteq [1, |S|]$ in S and $A^T \subseteq [1, |T|]$ in T , as summarised in the following lemma.

► **Lemma 21.** *Consider an instance of the k -LCS problem for $k = \mathcal{O}(1)$ and let $\ell \in [1, n]$. In $\mathcal{O}(n)$ time, one can construct sets $A^S \subseteq [1, |S|]$ and $A^T \subseteq [1, |T|]$ of size $\mathcal{O}(\frac{n}{\ell})$ satisfying the following condition: If a k -LCS of S and T has length $\ell' \in (\ell/2, \ell]$, then there exist positions $i^S \in [1, |S|]$, $i^T \in [1, |T|]$ and a shift $\delta \in [0, \ell')$ such that $i^S + \delta \in A^S$, $i^T + \delta \in A^T$, and the Hamming distance between $S[i^S \dots i^S + \ell']$ and $T[i^T \dots i^T + \ell']$ is at most k .*

Proof. As in [26], we say that position a in a string X is a *misperiod* with respect to a substring $X[i \dots j]$ if $X[a] \neq X[b]$, where b is the unique position such that $b \in [i, j]$ and $(j-i) \mid (b-a)$; for example, $j-i$ is a period of X if and only if there are no misperiods with respect to $X[i \dots j]$. We define the set $\text{LeftMisper}_k(X, i, j)$ as the set of k maximal misperiods that are smaller than i and $\text{RightMisper}_k(X, i, j)$ as the set of k minimal misperiods that are not smaller than j . Either set may have fewer than k elements if the corresponding misperiods do not exist. Further, let us define $\text{Misper}_k(X, i, j) = \text{LeftMisper}_k(X, i, j) \cup \text{RightMisper}_k(X, i, j)$ and $\text{Misper}(X, i, j) = \bigcup_{k=0}^{\infty} \text{Misper}_k(X, i, j)$.

Similar to Lemma 15, we construct three subsets of positions in $Y = \#S\$T$, where $\#, \$ \notin \Sigma$. For $\tau = \lfloor \ell/(6(k+1)) \rfloor$, let A_I be a τ -synchronising set of Y . Let $Y[i \dots j]$ be a τ -run with period p and assume that the first occurrence of its Lyndon root is at a position q of Y . Then, for $Y[i \dots j]$, for each $x \in \text{LeftMisper}_{k+1}(Y, i, i+p)$, we insert to A_{II} the two smallest positions in $[x+1, |Y|]$ that are equivalent to $q \pmod{p}$. Moreover, we insert to A_{III} the positions in $\text{Misper}_{k+1}(Y, i, i+p)$. Finally, we denote $A = A_I \cup A_{II} \cup A_{III}$, as well as $A^S = \{a-1 : a \in A \cap [2, |S|+1]\}$ and $A^T = \{a-|S|-2 : a \in A \cap [|S|+3, |Y|]\}$. The proof of the following claim, that can be found in the full version of this paper, resembles that of Lemma 15.

► **Claim 22.** The sets A^S and A^T satisfy the condition stated in Lemma 21.

It remains to show that the sets A^S and A^T can be constructed efficiently. A τ -synchronising set can be computed in $\mathcal{O}(n)$ time by Theorem 7 and all the τ -runs, together with the position of the first occurrence of their Lyndon root, can be computed in $\mathcal{O}(n)$ time [8]. After an $\mathcal{O}(n)$ -time preprocessing, for every τ -run, we can compute the set of the $k+1$ misperiods of its period to either side in $\mathcal{O}(1)$ time; see [26, Claim 18]. \blacktriangleleft

The next step in our solutions to long LCS and medium-length LCS was to construct an instance of the TWO STRING FAMILIES LCP PROBLEM. To adapt this approach, we generalise the notions of LCP and maxPairLCP so that they allow for mismatches. By $\text{LCP}_k(U, V)$, for $k \in \mathbb{Z}_{\geq 0}$, we denote the maximum length ℓ such that $U[1.. \ell]$ and $V[1.. \ell]$ are at Hamming distance at most k .

► **Definition 23.** *Given two sets $\mathcal{U}, \mathcal{V} \subseteq \Sigma^* \times \Sigma^*$ and two integers $k_1, k_2 \in \mathbb{Z}_{\geq 0}$, we define $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V}) = \max\{\text{LCP}_{k_1}(U_1, V_1) + \text{LCP}_{k_2}(U_2, V_2) : (U_1, U_2) \in \mathcal{U}, (V_1, V_2) \in \mathcal{V}\}$.*

Note that $\text{maxPairLCP}(\mathcal{U}, \mathcal{V}) = \text{maxPairLCP}_{0,0}(\mathcal{U}, \mathcal{V})$.

By Lemma 21, if a k -LCS of S and T has length $\ell' \in (\ell/2, \ell]$, then

$$\ell' = \max_{k'=0}^k \text{maxPairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V}), \text{ for } \mathcal{U} = \{((S[a - \ell \dots a])^R, S[a \dots a + \ell]) : a \in A^S\},$$

$$\mathcal{V} = \{((T[a - \ell \dots a])^R, T[a \dots a + \ell]) : a \in A^T\}.$$

Here, k' bounds the number of mismatches between $S[i^S \dots i^S + \delta)$ and $T[i^T \dots i^T + \delta)$, whereas $k - k'$ bounds the number of mismatches between $S[i^S + \delta \dots i^S + \ell')$ and $T[i^T + \delta \dots i^T + \ell')$. The following theorem, whose full proof can be found in the full version of this paper, is the most technical part of our contribution, and allows for efficiently computing the values $\text{maxPairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V})$.

► **Theorem 24.** *Consider two (ℓ, ℓ) -families \mathcal{U}, \mathcal{V} of total size N consisting of pairs of substrings of a given length- n text. For any non-negative integers $k_1, k_2 = \mathcal{O}(1)$, the value $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V})$ can be computed:*

- in $\mathcal{O}(n + N \log^{k_1+k_2+1} N)$ time and $\mathcal{O}(n + N)$ space if $\ell > \log^{3/2} N$,
- in $\mathcal{O}(n + N \ell \log^{k_1+k_2-1/2} N)$ time and $\mathcal{O}(n + N \ell / \log N)$ space if $\log N < \ell \leq \log^{3/2} N$,
- in $\mathcal{O}(n + N \ell^{k_1+k_2} \sqrt{\log N})$ time and $\mathcal{O}(n + N)$ space if $\ell \leq \log N$.

Proof Outline. We reduce the computation of $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V})$ into multiple computations of $\text{maxPairLCP}(\mathcal{U}', \mathcal{V}')$ across a family \mathbf{P} of pairs $(\mathcal{U}', \mathcal{V}')$ with $\mathcal{U}', \mathcal{V}' \subseteq \Sigma^* \times \Sigma^*$. Each pair $(U'_1, U'_2) \in \mathcal{U}'$ is associated to a pair $(U_1, U_2) \in \mathcal{U}$, with the string U'_i represented as a pointer to the source U_i and up to k_i substitutions needed to transform U_i to U'_i . Similarly, each pair $(V'_1, V'_2) \in \mathcal{V}'$ consists of *modified strings* with sources $(V_1, V_2) \in \mathcal{V}$. In order to guarantee $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V}) = \max_{(\mathcal{U}', \mathcal{V}') \in \mathbf{P}} \text{maxPairLCP}(\mathcal{U}', \mathcal{V}')$, we require $\text{LCP}(U'_i, V'_i) \leq \text{LCP}_{k_i}(U_i, V_i)$ for every $(U'_1, U'_2) \in \mathcal{U}'$ and $(V'_1, V'_2) \in \mathcal{V}'$ with $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ and that, for every $(U_1, U_2) \in \mathcal{U}$ and $(V_1, V_2) \in \mathcal{V}$, there exists $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ with $(U'_1, U'_2) \in \mathcal{U}'$ and $(V'_1, V'_2) \in \mathcal{V}'$, with sources (U_1, U_2) and (V_1, V_2) , respectively, such that $\text{LCP}(U'_i, V'_i) = \text{LCP}_{k_i}(U_i, V_i)$. Our construction is based on a technique of [59] which gives an analogous family for two subsets of Σ^* (rather than $\Sigma^* \times \Sigma^*$) and a single threshold: We apply the approach of [59] to $\mathcal{U}_i = \{U_i : (U_1, U_2) \in \mathcal{U}\}$ and $\mathcal{V}_i = \{V_i : (V_1, V_2) \in \mathcal{V}\}$ with threshold k_i , and then combine the two resulting families \mathbf{P}_i to derive \mathbf{P} .

Strengthening the arguments of [59], we show that each string $F_i \in \mathcal{U}_i \cup \mathcal{V}_i$ is the source of $\mathcal{O}(1)$ modified strings $F'_i \in \mathcal{U}'_i \cup \mathcal{V}'_i$ for any single $(\mathcal{U}'_i, \mathcal{V}'_i) \in \mathbf{P}_i$ and $\mathcal{O}(\min(\ell, \log N)^{k_i})$ modified strings across all $(\mathcal{U}'_i, \mathcal{V}'_i) \in \mathbf{P}_i$. This allows bounding the size of individual sets

$(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ by $\mathcal{O}(N)$ and the overall size by $\mathcal{O}(N \min(\ell, \log N)^{k_1+k_2})$. In order to efficiently build the compacted tries required at the input of the TWO STRING FAMILIES LCP PROBLEM, the modified strings $F'_i \in \mathcal{U}'_i \cup \mathcal{V}'_i$ are sorted lexicographically, and the two derived linear orders (for $i \in \{1, 2\}$) are maintained along with every pair $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$. Overall, the family \mathbf{P} is constructed in $\mathcal{O}(n + N \min(\ell, \log N)^{k_1+k_2})$ time and $\mathcal{O}(n + N)$ space.

The resulting instances of the TWO STRING FAMILIES LCP PROBLEM are solved using Lemma 3 (if $\ell > \log^{3/2} N$) or Lemma 4 otherwise; note that $\mathcal{U}', \mathcal{V}'$ are (ℓ, ℓ) -families. ◀

Recall that the algorithm of Theorem 24 is called $k + 1 = \mathcal{O}(1)$ times, always with $N = |A^S| + |A^T| = \mathcal{O}(n/\ell)$. Overall, the value $\max_{k'=0}^k \text{maxPairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V})$ is therefore computed in $\mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n)$ space in each of the following cases:

- in $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} N) = \mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n + \frac{n}{\ell}) = \mathcal{O}(n)$ space if $\ell > \log^{3/2} N$;
- in $\mathcal{O}(n + \frac{n}{\ell} \ell \log^{k-1/2} N) = \mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n + \frac{n}{\ell} \ell / \log N) = \mathcal{O}(n)$ space if $\log N < \ell \leq \log^{3/2} N$;
- in $\mathcal{O}(n + \frac{n}{\ell} \ell^k \sqrt{\log N}) = \mathcal{O}(n \log^{k-1} n)$ time and $\mathcal{O}(n + \frac{n}{\ell}) = \mathcal{O}(n)$ space if $\ell \leq \log N$.

Accounting for $\mathcal{O}(n)$ time and space to determine the length d of an LCS between S and T , and the $\mathcal{O}(\log k)$ values ℓ that need to be tested so that the intervals $(\ell/2, \ell]$ cover $[d, (k+1)d + k]$, this concludes the proof of Theorem 2.

Moreover, the three cases yield the following complexities: $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} n)$ if $\ell > \log^{3/2} N$, $\mathcal{O}(n \log^{k-1/2} n) = \mathcal{O}(\frac{n}{\ell} \log^{k+1} n)$ if $\log N < \ell \leq \log^{3/2} N$, and $\mathcal{O}(n \log^{k-1} n) = \mathcal{O}(\frac{n}{\ell} \log^{k+1} n)$ if $\ell \leq \log N$, which gives an $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} n)$ -time solution for any ℓ , thus improving [24] for $k = \mathcal{O}(1)$.

References

- 1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.
- 2 Amihood 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 für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.11.
- 3 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/s00453-020-00744-0.
- 4 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.
- 5 Maxim A. Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. 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 572–591. SIAM, 2015. doi:10.1137/1.9781611973730.39.
- 6 Maxim A. Babenko and Tatiana Starikovskaya. Computing the longest common substring with one mismatch. *Problems of Information Transmission*, 47(1):28–33, 2011. doi:10.1134/S0032946011010030.

- 7 Ricardo A. Baeza-Yates. Improved string searching. *Software: Practice and Experience*, 19(3):257–271, 1989. doi:10.1002/spe.4380190305.
- 8 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 9 Djamal Belazzougui. Worst case efficient single and multiple string matching in the RAM model. In Costas S. Iliopoulos and William F. Smyth, editors, *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers*, volume 6460 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2010. doi:10.1007/978-3-642-19222-7_10.
- 10 Djamal Belazzougui. Improved space-time tradeoffs for approximate full-text indexing with one edit error. *Algorithmica*, 72(3):791–817, 2015. doi:10.1007/s00453-014-9873-9.
- 11 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Optimal packed string matching. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 423–432. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.423.
- 12 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Towards optimal packed string matching. *Theoretical Computer Science*, 525:111–129, 2014. doi:10.1016/j.tcs.2013.06.013.
- 13 Stav Ben-Nun, Shay Golan, Tomasz Kociumaka, and Matan Kraus. Time-space tradeoffs for finding a long common substring. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.5.
- 14 Philip Bille. Fast searching in packed strings. In Gregory Kucherov and Esko Ukkonen, editors, *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Lille, France, June 22-24, 2009, Proceedings*, volume 5577 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2009. doi:10.1007/978-3-642-02441-2_11.
- 15 Philip Bille. Fast searching in packed strings. *Journal of Discrete Algorithms*, 9(1):49–56, 2011. doi:10.1016/j.jda.2010.09.003.
- 16 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 6:1–6:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.6.
- 17 Dany Breslauer, Leszek Gasieniec, and Roberto Grossi. Constant-time word-size string matching. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings*, volume 7354 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 2012. doi:10.1007/978-3-642-31265-6_7.
- 18 Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. 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 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69.
- 19 Gerth Stølting Brodal and Christian N. S. Pedersen. Finding maximal quasiperiodicities in strings. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2000. doi:10.1007/3-540-45123-4_33.

- 20 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *Journal of the ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.
- 21 Domenico Cantone and Simone Faro. Pattern matching with swaps for short patterns in linear time. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings*, volume 5404 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2009. doi:10.1007/978-3-540-95891-8_25.
- 22 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. Compressed indexes for approximate string matching. *Algorithmica*, 58(2):263–281, 2010. doi:10.1007/s00453-008-9263-2.
- 23 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. A linear size index for approximate pattern matching. *Journal of Discrete Algorithms*, 9(4):358–364, 2011. doi:10.1016/j.jda.2011.04.004.
- 24 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 für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.23.
- 25 Panagiotis Charalampopoulos, Paweł Gawrychowski, and Karol Pokorski. Dynamic longest common substring in polylogarithmic time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.27.
- 26 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba. Circular pattern matching with k mismatches. *Journal of Computer and System Sciences*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.
- 27 Archie L. Cobbs. Fast approximate matching using suffix trees. In Zvi Galil and Esko Ukkonen, editors, *Combinatorial Pattern Matching, 6th Annual Symposium, CPM 95, Espoo, Finland, July 5-7, 1995, Proceedings*, volume 937 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 1995. doi:10.1007/3-540-60044-2_33.
- 28 Vincent Cohen-Addad, Laurent Feuilloley, and Tatiana Starikovskaya. Lower bounds for text indexing with mismatches and differences. 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 1146–1164. SIAM, 2019. doi:10.1137/1.9781611975482.70.
- 29 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004. doi:10.1145/1007352.1007374.
- 30 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
- 31 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.
- 32 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.63.

- 33 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.
- 34 Nathan J. Fine and Herbert 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>.
- 35 Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643–647, 2015. doi:10.1016/j.ipl.2015.03.006.
- 36 Kimmo Fredriksson. Faster string matching with super-alphabets. In Alberto H. F. Laender and Arlindo L. Oliveira, editors, *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings*, volume 2476 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2002. doi:10.1007/3-540-45735-6_5.
- 37 Kimmo Fredriksson. Shift-or string matching with super-alphabets. *Information Processing Letters*, 87(4):201–204, 2003. doi:10.1016/S0020-0190(03)00296-5.
- 38 François Le Gall and Saeed Seddighin. Quantum meets fine-grained complexity: Sublinear time quantum algorithms for string problems. *CoRR*, abs/2010.12122, 2020. arXiv:2010.12122.
- 39 Emanuele Giaquinta, Szymon Grabowski, and Kimmo Fredriksson. Approximate pattern matching with k -mismatches in packed text. *Information Processing Letters*, 113(19-21):693–697, 2013. doi:10.1016/j.ipl.2013.07.002.
- 40 Szymon Grabowski. A note on the longest common substring with k -mismatches problem. *Information Processing Letters*, 115(6-8):640–642, 2015. doi:10.1016/j.ipl.2015.03.003.
- 41 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 841–850. ACM/SIAM, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644250>.
- 42 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.
- 43 Lucas Chi Kwong Hui. Color set size problem with application to string matching. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 1992. doi:10.1007/3-540-56024-6_19.
- 44 Trinh N. D. Huynh, Wing-Kai Hon, Tak Wah Lam, and Wing-Kin Sung. Approximate string matching using compressed suffix arrays. *Theoretical Computer Science*, 352(1-3):240–249, 2006. doi:10.1016/j.tcs.2005.11.022.
- 45 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.
- 46 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 47 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.
- 48 Shmuel Tomi Klein and Miri Ben-Nissan. Accelerating Boyer Moore searches on binary texts. In Jan Holub and Jan Zdárek, editors, *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers*, volume 4783 of *Lecture Notes in Computer Science*, pages 130–143. Springer, 2007. doi:10.1007/978-3-540-76336-9_14.

- 49 Tomasz Kociumaka, Jakub Radoszewski, and Tatiana Starikovskaya. Longest common substring with approximately k mismatches. *Algorithmica*, 81(6):2633–2652, 2019. doi:10.1007/s00453-019-00548-x.
- 50 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wrocław, Poland, September 8-10, 2014. Proceedings*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.
- 51 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. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.
- 52 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
- 53 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPICs*, pages 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CPM.2020.24.
- 54 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.
- 55 Gonzalo Navarro and Mathieu Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In Martin Farach-Colton, editor, *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 20-22, 1998, Proceedings*, volume 1448 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1998. doi:10.1007/BFb0030778.
- 56 Tatiana 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.
- 57 Jorma Tarhio and Hannu Peltola. String matching in the DNA alphabet. *Software: Practice and Experience*, 27(7):851–861, 1997.
- 58 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.
- 59 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.
- 60 Dekel Tsur. Fast index for approximate string matching. *Journal of Discrete Algorithms*, 8(4):339–345, 2010. doi:10.1016/j.jda.2010.08.002.
- 61 Esko Ukkonen. Approximate string-matching over suffix trees. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, 4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993, Proceedings*, volume 684 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 1993. doi:10.1007/BFb0029808.
- 62 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.

Feature Cross Search via Submodular Optimization

Lin Chen¹ ✉

Simons Institute for the Theory of Computing, University of California, Berkeley, CA, USA

Hossein Esfandiari ✉

Google Research, New York, NY, USA

Gang Fu ✉

Google Research, New York, NY, USA

Vahab S. Mirrokni ✉

Google Research, New York, NY, USA

Qian Yu ✉

Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, USA

Abstract

In this paper, we study feature cross search as a fundamental primitive in feature engineering. The importance of feature cross search especially for the linear model has been known for a while, with well-known textbook examples. In this problem, the goal is to select a small subset of features, combine them to form a new feature (called the crossed feature) by considering their Cartesian product, and find feature crosses to learn an *accurate* model. In particular, we study the problem of maximizing a normalized Area Under the Curve (AUC) of the linear model trained on the crossed feature column.

First, we show that it is not possible to provide an $n^{1/\log \log n}$ -approximation algorithm for this problem unless the exponential time hypothesis fails. This result also rules out the possibility of solving this problem in polynomial time unless $P = NP$. On the positive side, by assuming the naïve Bayes assumption, we show that there exists a simple greedy $(1 - 1/e)$ -approximation algorithm for this problem. This result is established by relating the AUC to the total variation of the commutator of two probability measures and showing that the total variation of the commutator is monotone and submodular. To show this, we relate the submodularity of this function to the positive semi-definiteness of a corresponding kernel matrix. Then, we use Bochner's theorem to prove the positive semi-definiteness by showing that its inverse Fourier transform is non-negative everywhere. Our techniques and structural results might be of independent interest.

2012 ACM Subject Classification Computing methodologies → Feature selection

Keywords and phrases Feature engineering, feature cross, submodularity

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.31

Related Version *Full Version:* <https://arxiv.org/pdf/2107.02139.pdf>

1 Introduction

Feature engineering is one of the most fundamental problems in machine learning and it is the key to all supervised learning models. In feature engineering, we start with a collection of features (a.k.a., raw attributes) and turn them into a new set of features, with the purpose of improving the accuracy of the learning model. This is often done by some basic operations, such as removing irrelevant and redundant features (studied as feature selection [10, 32, 33, 26, 11, 25, 18]), combining features (a.k.a., feature cross [30, 20]) and bucketing and compressing the vocabulary of the features [2, 7, 28, 1].

¹ Authors are ordered alphabetically.



© Lin Chen, Hossein Esfandiari, Gang Fu, Vahab S. Mirrokni, and Qian Yu;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 31; pp. 31:1–31:16

Leibniz International Proceedings in Informatics



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

Finding an *efficient* set of features to combine (*i.e.*, cross) is one of the main primitives in feature engineering. Let us start with a text book example to show the importance of feature cross for the linear model. Consider a model with two features, language, which can be English or Spanish, and country, which can be Mexico or Scotland. Say if English appears with Scotland, or if Spanish appears with Mexico, the label is 1. Otherwise the label is 0. It is easy to see that in this case there is no linear model using these two features with a nontrivial accuracy (*i.e.*, the best model matches the label with probability $1/2$). By crossing these two features, we get a new feature with four possible values (English, Mexico), (English, Scotland), (Spanish, Mexico), (Spanish, Scotland). Now, a linear model based on this new feature can perfectly match the label. This is a well-known concept in feature engineering.

Unlike feature selection and vocabulary compression, and despite the importance of feature cross search in practice, this problem is not well studied from a theoretical perspective. While some heuristics and exponential-time algorithms have been developed for this problem (*e.g.*, [30, 20]), the complexity of designing approximation algorithms for this problem is not studied. This might be due to the complex behavior of crossing features on the accuracy of the learning models. In this work, we provide a simple formulation of this problem, and initiate a theoretical study.

Let us briefly define the problem as follows and defer the formal definition of the problem to a later section: Given a set of n features, and a number k , compute a set of at most k features out of n features and combine these k features such that the accuracy of the optimum linear model on the combined feature is maximized. To measure the accuracy we use normalized *Area Under the Curve* (AUC). The bound k is to avoid over fitting.² This is a very basic definition for the feature cross search problem and can be considered as a building block in feature engineering. In fact, as we discuss later in the paper, it is still hard to design algorithms for this basic problem.

First, we show that there is no $n^{1/\log \log n}$ -approximation algorithm for feature cross search unless the exponential time hypothesis fails. Our hardness result also implies that there does not exist a polynomial-time algorithm for feature cross search unless $P = NP$. It is easy to extend these hardness results to other notions of accuracy such as probability of matching the label. Obviously, this hardness result holds for any extension of the problem as well.

In fact, often, the real world inputs are not adversarially constructed. Usually, the inputs follow some structural properties that allow simple algorithms to work efficiently. With this intuition in mind, to complement our hardness result, for features under the naïve Bayes assumption [22, 29, 9], we provide a $(1 - 1/e)$ -approximation algorithm that only needs polynomially many function evaluations. We further discuss and justify this assumption in Section 1.1.

In Section 1.1, we define the problem formally and present our results as well as an overview of our techniques. In Section 2, we provide the preliminary definitions and observations that will be used later in the proofs. In Section 3, we present our hardness results. We relate the maximum AUC to the log-likelihood ratio and the total variation of the commutator of two probability distributions in Section 4. Section 5 establishes the monotonicity and submodularity of the maximum AUC as a set function. This section forms the most technical part of the paper. In Section 6 we present other related works. Finally, Section 7 concludes the paper.

² In practice this number is chosen by tracking the accuracy of the model on the validation data. However, this is out of the scope of this paper.

1.1 Problem Statement and Our Contributions

We start with some definitions necessary to present our results, and then, we present our contributions. Assume that the dataset comprises $n = |U|$ categorical feature columns and a binary label column, where U is the set of all feature columns. Let the random variable X_i denote the value of the i -th feature column ($i \in U$) and $C \in \{0, 1\}$ be the value of the binary label. The random variables $X_1, \dots, X_{|U|}, C$ follow a joint distribution \mathcal{D} . Additionally, we assume that the support of the random variable X_i is a finite set $V_i \subseteq \mathbb{N}$. The set V_i is also known as the *vocabulary* of the i -th feature column. If $A \subseteq U$ is a set of feature columns, we write V_A for $\prod_{a \in A} V_a$ and write X_A for $(X_a | a \in A)$, where $(X_a | a \in A)$ is a vector indexed by $a \in A$ (for example, if $A = \{1, 2, 4\}$, the vector X_A is a 3-dimensional vector (X_1, X_2, X_4)).

Suppose that we focus on a set of feature columns and temporarily ignore the remaining feature columns. In other words, we consider the dataset modeled by the distribution of (X_A, C) . Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ denote the set of extended real numbers. Given a function $\sigma : V_A \rightarrow \overline{\mathbb{R}}$ that assigns a score to each possible value of X_A , and given a threshold τ , we predict a positive label for X_A if $\sigma(X_A) > \tau$ and predict a negative label if $\sigma(X_A) < \tau$. If $\sigma(X_A) = \tau$, we allow for predicting a positive or negative label at random. Let TPR and FPR denote the true and false positive rate of this model given a certain decision rule, respectively. Note that both true and false positive rates lie in $[0, 1]$. If one varies τ from $-\infty$ to ∞ while fixing the score function σ , a curve that consists of the collection of achievable points (FPR, TPR) is produced and the curve resides in the square $[0, 1] \times [0, 1]$. The area under the curve (AUC) [4] is then defined as the area of the region enclosed by this curve, and the two lines FPR = 1 and TPR = 0.

An equivalent definition is that AUC is roughly the probability that a random positive instance has a higher score (in terms of σ) than a negative instance (we say *roughly* because in Definition 1, we have to be careful about tiebreaking, i.e., the second term).

► **Definition 1** (Area under the curve (AUC) [4]). *Given a set of feature columns A and a function $\sigma : V_A \rightarrow \mathbb{R}$, the area under the curve (AUC) of A and σ is*

$$\text{AUC}_\sigma(A) = \Pr[\sigma(X_A^+) > \sigma(X_A^-) | C^+ = 1, C^- = 0] + \frac{1}{2} \Pr[\sigma(X_A^+) = \sigma(X_A^-) | C^+ = 1, C^- = 0],$$

where $(X_A^+, C^+), (X_A^-, C^-) \sim \mathcal{D}$ are i.i.d. and $X_A^\gamma = (X_a^\gamma | a \in A)$ obeys a marginal distribution of \mathcal{D} (γ is either $+$ or $-$).

The maximum AUC is the AUC of the best scoring function. It is a function of the set of feature columns and independent of the scoring function.

► **Definition 2** (Maximum AUC). *Given a set of feature columns A , the maximum AUC is*

$$\text{AUC}^*(A) = \sup_{\sigma: V_A \rightarrow \mathbb{R}} \text{AUC}_\sigma(A).$$

Now we are ready to present our results. We start with Observation 3 which provides a characterization of AUC via the total variation distance.

► **Observation 3** (AUC as total variation distance). *Let P_i^A be the conditional distribution $\Pr[X_A = i]$ on V_A and let $d_{TV}(P, Q)$ denote the total variation distance between two probability measures P and Q . We have*

$$\text{AUC}^*(A) = \frac{1}{2} + \frac{1}{2} d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A) = \frac{1}{2} + \frac{1}{2} \sum_{x, y \in V_A} |P_1^A(x)P_0^A(y) - P_0^A(x)P_1^A(y)|,$$

where $P_1^A \times P_0^A$ and $P_0^A \times P_1^A$ denote the product measures.

Recall that if P and Q are two probability measures on a common σ -algebra \mathcal{F} , the *total variation distance* between them is

$$d_{TV}(P, Q) \triangleq \sup_{A \in \mathcal{F}} |P(A) - Q(A)| \in [0, 1].$$

If the sample space Ω (the set of all outcomes) is finite, Scheffé's lemma [27] gives

$$d_{TV}(P, Q) = \frac{1}{2} \|P - Q\|_1 \triangleq \frac{1}{2} \sum_{\omega \in \Omega} |P(\omega) - Q(\omega)|. \quad (1)$$

We present the proof of Observation 3 in Section 4. Observation 3 shows that the maximum AUC is an affine function of the total variation distance between $P_1^A \times P_0^A$ and $P_0^A \times P_1^A$, where P_i^A is the probability measure conditioned on the label. In light of (1), we have $d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A) = \frac{1}{2} \|P_1^A \times P_0^A - P_0^A \times P_1^A\|_1$. We call the signed measure $P_1^A \times P_0^A - P_0^A \times P_1^A$ the *commutator* of the two probability measures P_0^A and P_1^A . Our second remark is that since the total variation distance always resides on $[0, 1]$, the range of the maximum AUC is $[1/2, 1]$.

The next theorem is our main hardness result, stating that it is not possible to approximate feature cross search, unless the exponential time hypothesis [12] fails. We consider maximization of $2 \text{AUC}^*(A) - 1$ rather than $\text{AUC}^*(A)$ in (2) because the range of the maximum AUC is $[1/2, 1]$ (as we remark before) and assigning the same score to all feature values in V_A attains an AUC of $1/2$, thereby achieving at least a $1/2$ -approximation. In light of its range, we consider its normalized version $2 \text{AUC}^*(A) - 1$ whose range is $[0, 1]$. We prove this theorem in Section 3. In fact, our hardness result also implies that the feature cross search problem is NP-hard (see Corollary 14).

► **Theorem 4.** *There is no $n^{1/\text{poly}(\log \log n)}$ -approximation algorithm for the following maximization problem unless the exponential time hypothesis [12] fails.*

$$\max_{A \subseteq U, |A|=k} (2 \text{AUC}^*(A) - 1). \quad (2)$$

Although the above hardness result rules out the existence of an algorithm with a good approximation factor in the general case, it is very rare to face such hard examples in practice. We consider the naïve Bayes assumption that all feature columns are conditionally independent given the label. We borrowed this assumption from the widely-used naïve Bayes classifier [22]. For example, under the same assumption, [15] established the submodularity of mutual information and [6] proved that in the sequential information maximization problem, the most informative selection policy behaves near optimally. [29] conducted an empirical study on the public software defect data from NASA with PCA pre-processing. They concluded that this assumption was not harmful. Although relaxing the assumption could produce numerically more favorable results, they were not statistically significantly better than assuming this assumption. In another example [9], based on their analysis on three real-world datasets for natural language processing tasks (MDR, Newsgroup and the ModApte version of the Reuters-21578), they drew a similar conclusion that relaxing the assumption did not improve the performance.

► **Assumption 1 (Naïve Bayes).** Given the label, all feature columns are independent. In other words, it holds for $A \subseteq U$ and $i = 0, 1$ that

$$\Pr[X_A = x_A | C = i] = \prod_{a \in A} \Pr[X_a = x_a | C = i]. \quad (3)$$

Our major algorithmic contribution is to show that under the naïve Bayes assumption the set function AUC^* is monotone submodular, which in turn, implies that a greedy algorithm provides a constant-factor approximation algorithm for this problem.³

► **Theorem 5.** *Under the naïve Bayes assumption, the set function $\text{AUC}^* : 2^U \rightarrow \mathbb{R}$ is monotone submodular.*

This theorem implies the following result in light of the result of [24].

► **Corollary 6.** *Under the naïve Bayes assumption, there exists a $(1 - 1/e)$ -approximation algorithm that only needs polynomially many evaluations of AUC^* for feature cross search.*

To show Theorem 5, we prove Proposition 7. Proving this proposition requires an involved analysis and it may be of independent interest in statistics.

► **Proposition 7** (Proof in Section 5). *Let U be a finite index set. Assume that for every $a \in U$, there are a pair of probability measures P_0^a and P_1^a on a common sample space V_a . For any $A \subseteq U$, define the set function $F : 2^U \rightarrow \mathbb{R}_{\geq 0}$ by*

$$F(A) = d_{TV} \left(\prod_{a \in A} P_1^a \times \prod_{a \in A} P_0^a, \prod_{a \in A} P_0^a \times \prod_{a \in A} P_1^a \right). \quad (4)$$

The set function F is monotone and submodular.

Its proof is presented in Section 5. In fact, the most technical part of this paper is to prove Proposition 7 which claims that the total variation of the commutator of probability measures is monotone submodular. The monotonicity is a consequence of the subadditivity of the absolute value function. Submodularity is the technically harder part and is shown in the following four steps.

First, we introduce the notion of *involution equivalence*. An involution is a map from a set to itself that is equal to its inverse map. Two probability measures P and P' are said to be involution equivalent if there exists an involution f on the sample space Ω such that for every $x \in \Omega$, it holds that $P(x) = P'(f(x))$. Note that if $P(x) = P'(f(x))$ holds for every $x \in \Omega$, we have $P'(x) = P(f(x))$ also holds for every $x \in \Omega$. In fact, it defines a symmetric relation on probability measures on Ω . If P and Q are two probability measures on a common sample space, the product measures $P \times Q$ and $Q \times P$ are involution equivalent and connected by the natural transpose involution f that sends $(x, y) \in \Omega^2$ to $(y, x) \in \Omega^2$.

The second step is in light of a key observation that summing a bivariate function of two involution equivalent probability measures over the common sample space remains invariant under the swapping of the two measures. Based on this key observation, if P and P' are involution equivalent, for every x in their common sample space, we construct the probability measures of two Bernoulli random variables U_x and U'_x such that $U_x(1) = U'_x(0) = \frac{P(x)}{P(x) + P'(x)}$ and $U_x(0) = U'_x(1) = \frac{P'(x)}{P(x) + P'(x)}$. The two Bernoulli probability measures U_x and U'_x are again involution equivalent and connected by the swapping of 0 and 1. To establish submodularity, one has to check an inequality that characterizes the diminishing returns property (see equation (5) in Section 2.2). Another key observation is that after defining the Bernoulli probability measures, the desired inequality can be shown to be a conic combination of the same inequality with *some* (not all) probability measures in the inequality replaced

³ We will review the definition of submodular and monotone set functions in Section 2.2.

by the Bernoulli probability measures U_x and U'_x . To make the above observation work, we have to require that the remaining probability measures unreplaced in the inequality must be either of the form $P \times Q$ or its transpose $Q \times P$. Using this approach, we reduce the problem to the Bernoulli case.

Third, after reducing the problem to the Bernoulli case, performing a series of more involved algebraic manipulations, we re-parametrize the desired inequality that formulates the diminishing returns property and obtain that the inequality is equivalent to the positive semi-definiteness of a quadratic form with respect to a kernel matrix. However, this re-parametrization is valid only for elements of a positive measure with respect to some probability measures in the inequality. As a consequence, prior to the algebraic manipulations and re-parametrization, we have to eliminate those elements of measure zero by showing that their total contribution to the sum is zero. We would like to remark here that the individual terms may not be zero but they are canceled out under the summation.

Finally, to show that the aforementioned kernel matrix is positive semi-definite, we prove that it is induced by a positive definite function. We establish the positive definiteness of the function by showing that its inverse Fourier transform is non-negative everywhere (this is an implication of the Bochner's theorem, see Section 2).

Theorem 5 is a straightforward corollary of Proposition 7.

Proof. Let $P_i^A[\cdot]$ denote $\Pr[X_A|C = i]$, the conditional probability measure on V_A given the labeling being i , where $i = 0, 1$. When $A = \{a\}$ is a singleton, we write P_i^a for $P_i^{\{a\}}$ as a shorthand notation. Under Assumption 1, (3) can be re-written as $P_i^A[x_A] = \prod_{a \in A} P_i^a[x_a]$, or in a more compact way,

$$P_i^A = \bigotimes_{a \in A} P_i^a.$$

By Observation 3, we have

$$\begin{aligned} \text{AUC}^*(A) &= \frac{1}{2} + \frac{1}{2} d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A) \\ &= \frac{1}{2} + \frac{1}{2} d_{TV}\left(\bigotimes_{a \in A} P_1^a \times \bigotimes_{a \in A} P_0^a, \bigotimes_{a \in A} P_0^a \times \bigotimes_{a \in A} P_1^a\right) \\ &= \frac{1}{2} + \frac{1}{2} F(A). \end{aligned}$$

Since $F(A)$ is monotone submodular by Proposition 7, so is AUC^* . ◀

2 Preliminaries

Throughout this paper, let Δ_Ω denote the set of all probability measures on a finite set Ω and we always assume that the sample space Ω is finite. The set of extended real numbers is denoted by $\overline{\mathbb{R}}$ and defined as $\mathbb{R} \cup \{-\infty, +\infty\}$.

2.1 Involution Equivalence

We first review the definition of an involution.

► **Definition 8** (Involution). *A map $f : \Omega \rightarrow \Omega$ is said to be an involution if for all $x \in \Omega$, it holds that $f(f(x)) = x$.*

In this paper, we introduce a new notion termed *involution equivalence*, which forms an equivalence relation on Δ_Ω . Intuitively, two probability measures on a common sample space Ω are involution equivalent if they are the same after renaming the elements in Ω via an involution.

► **Definition 9** (Involution equivalence). *Let P, P' be two probability measures on a common sample space Ω . We say that P and P' are involution equivalent if there exists an involution f such that for all $x \in \Omega$, $P(x) = P'(f(x))$. If P and P' are involution equivalent, we denote it by $P \stackrel{f}{\sim} P'$ or $P \sim P'$ with the involution f omitted when it is not of our interest.*

► **Remark 10.** If $P \stackrel{f}{\sim} P'$, we have $P'(x) = P'(f(f(x))) = P(f(x))$.

► **Remark 11** (Transpose involution). If P and P' are two probability measures on a common sample space Ω , the product measure $P \times P'$ is involution equivalent to $P' \times P$ via the natural transpose map \top that sends $(x, y) \in \Omega^2$ to $\top(x, y) = (y, x) \in \Omega^2$. Thus we write $P \times P' \stackrel{\top}{\sim} P' \times P$ and term \top a *transpose involution*.

2.2 Submodular and Monotone Set Functions

Let us recall the definition of submodular and monotone set functions. Submodular set functions are those satisfying that the marginal gain of adding a new element to a set is no smaller than that of adding the same element to its superset. This property is called the *diminishing returns property*, which naturally arises in data summarization [23], influence maximization [34], and natural language processing [19], among others.

► **Definition 12** (Submodular set function, [24, 14]). *A set function $f : 2^U \rightarrow \mathbb{R}_{\geq 0}$ is submodular if for any $A \subseteq U$ and $a, b \in U \setminus A$ such that $a \neq b$, it satisfies*

$$f(A \cup \{a\}) - f(A) \geq f(A \cup \{a, b\}) - f(A \cup \{b\}). \quad (5)$$

The above Equation (5) formulates the diminishing returns property. Its left-hand side is the marginal gain of adding a to a set A while the right-hand side is the marginal gain of adding the same element a to the superset $A \cup \{b\}$.

A monotone set function is a function that assigns a higher function value to a set than all its subsets.

► **Definition 13** (Monotone set function). *A set function $f : 2^U \rightarrow \mathbb{R}$ is monotone if for any $A \subseteq B \subseteq U$, we have $f(A) \leq f(B)$.*

3 Hardness Result

In this section we show the hardness of approximation of the feature cross search problem. We say an algorithm is an α -approximation algorithm for the feature cross search problem if its accuracy (i.e., $2 \text{AUC} - 1$) is at least α times that of the optimum algorithm.

As a byproduct, we show a hardness result for a feature selection problem based on mutual information defined as follows. In the label-based mutual information maximization problem we have a universe of features U and a vector of labels C , and we want to select a subset S of size k from U that maximizes the mutual information $I(S; C)$. In other words we want to solve $\arg\max_{S \subseteq U, |S|=k} I(S; C)$. We say an algorithm is an α -approximation algorithm for the label-based mutual information maximization problem if it reports a set S such that

$$\alpha \leq \frac{I(S; C)}{\max_{S' \subseteq U, |S'|=k} I(S'; C)}.$$

For both problems, we show that an α -approximation algorithm for the problem implies an α -approximation algorithm for the k -densest subgraph problem. In the k -densest subgraph problem we are given a graph $G(V, E)$ and a number k and we want to pick a subset S of size k from V such that the number of edges induced by S is maximized. Recently, [21] shows that there is no [almost polynomial] $n^{-1/\text{poly}(\log \log n)}$ -approximation algorithm for k -densest subgraph that runs in polynomial time unless the exponential time hypothesis fails. The best known algorithm for this problem has approximation factor $n^{-1/4}$ [3].

► **Theorem 4.** *There is no $n^{1/\text{poly}(\log \log n)}$ -approximation algorithm for the following maximization problem unless the exponential time hypothesis [12] fails.*

$$\max_{A \subseteq U, |A|=k} (2 \text{AUC}^*(A) - 1). \quad (2)$$

Proof. We prove this theorem via an approximation preserving reduction from k -densest subgraph. Let $G(V, E)$ be an instance of k -densest subgraph problem. We construct a set of features as follows. There are $n = |V|$ features each corresponding to one vertex of G . For a vertex $v \in V$ we indicate the value of the feature corresponding to v by x_v . There are three possible feature values, 0, 1 and $\#$. The values of the features are determined by the following random process. Select an edge (u, v) uniformly at random from E . The value of the features x_v and x_u are chosen independently and uniformly at random from $\{0, 1\}$. The value of all other features are $\#$. The value of the label is $x_v \oplus x_u$. To show the hardness of approximation of the feature cross search, we show that any solution of accuracy $\phi = 2 \text{AUC} - 1$ corresponds to a subgraph of G with k vertices and ϕm edges and vice versa.

Let H be a subgraph of G with k vertices and ϕm edges. Let S be the set of features corresponding to the vertices in H . We analyze this in two cases.

Case 1. The value of the crossed feature contains zero or one numbers (*i.e.*, all are $\#$, or all but one are $\#$). Note that this case corresponds to a scenario that the pair of features with binary value are not both in S and hence it happens with probability $\frac{m-\phi m}{m} = 1 - \phi$. Moreover, note that in this case the value of the crossed feature is independent of the value of the label (*i.e.*, given the value of the feature the label is 0 or 1 with probability $1/2$).

Case 2. The value of the crossed feature contains two numbers. In this case one can easily predict the correct label with probability 1 (*i.e.*, if the numbers are both 0 or both 1 output 0, otherwise output 1). Moreover, note that this case corresponds to a scenario that the pair of features with binary values are both in S and hence it happen with probability $\frac{\phi m}{m} = \phi$.

Case 1 happens with probability $1 - \phi$ and in this case the label is independent of the value of the crossed feature, and Case 2 happens with probability ϕ , where the label can be predicted with probability 1. Therefore, we have $\text{AUC} = \int_0^1 \phi + (1 - \phi)p dp = \phi + \frac{1-\phi}{2} = \frac{1+\phi}{2}$ which gives us $2 \text{AUC} - 1 = \phi$ as claimed. ◀

In fact, the densest subgraph problem is NP-hard as well, and hence the reduction in the proof of Theorem 4 directly implies the NP-hardness of feature cross search as well.

► **Corollary 14.** *The feature cross search problem is NP-hard.*

Similar proof to that of Theorem 4 implies the hardness of feature selection via label based mutual information maximization.

► **Theorem 15.** *There is no $n^{-1/\text{poly}(\log \log n)}$ -approximation algorithm for feature selection via label based mutual information maximization unless the exponential time hypothesis fails.*

Proof. Similar to Theorem 4 we prove this theorem via an approximation preserving reduction from k -densest subgraph. Consider the hard example provided in the proof of Theorem 4. Here we show that for any arbitrary set of features S if the induced subgraph of the corresponding vertices has ϕm edges, we have $I(S; C) = \phi$. We define a random variable X as follows. X is 0 if none or one of the features in S has a binary value, and X is 1 if two of the features in S have binary values. Note that the value of C is independent of X , and thus we have $I(S; C) = I(S; C|X)$. Hence, we have

$$\begin{aligned} I(S; C) &= I(S; C|X) = \mathbb{E}_X[I(S, C)|X] \\ &= \Pr[X = 0](I(S; C)|X = 0) + \Pr[X = 1](I(S; C)|X = 1). \end{aligned}$$

Note that given $X = 0$, S and C are independent and hence we have $(I(S; C)|X = 0) = 0$. On the other hand if $X = 1$, S uniquely defines C , and hence we have $(I(S; C)|X = 1) = 1$. Therefore we have $I(S; C) = \mathbb{E}_X[I(S, C)|X] = \Pr[X = 1] = \frac{\phi m}{m} = \phi$, as claimed. ◀

4 Reformulating Maximum AUC

Here, we prove Observation 3 and thereby reformulate the maximum AUC as an affine function of the total variation of the commutator of two probability measures. Furthermore, we show that the maximum AUC is achieved by a specific scoring function, *i.e.*, the log-likelihood ratio.

We start with some definitions. We define the log-likelihood ratio of an event E by $\mathcal{L}(E) = \log \frac{P_1[E]}{P_0[E]}$ provided that $P_0[E]P_1[E] \neq 0$, where $P_i[\cdot] = \Pr[\cdot|C = i]$. If $P_0[E] = 0$, the log-likelihood ratio $\mathcal{L}(E)$ is defined to be $+\infty$. If $P_1[E] = 0$, it is defined to be $-\infty$. As a result, the range of the log-likelihood ratio is the set of extended real numbers, denoted by $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. Proposition 16 shows that the maximum AUC is achieved by a specific scoring function, *i.e.*, the log-likelihood ratio \mathcal{L} . Here we abuse the notation and define the score $\mathcal{L}(x_A)$ assigned to each $x_A \in V_A$ to be $\mathcal{L}(X_A = x_A)$, where X_A and C are jointly sampled from \mathcal{D} . In other words, if we assign to each value in V_A a score accordingly, then the AUC is maximized.

► **Proposition 16.** *The log-likelihood ratio achieves the maximum AUC among all functions $\sigma : V_A \rightarrow \overline{\mathbb{R}}$*

$$\text{AUC}_{\mathcal{L}}(A) = \max_{\sigma: V_A \rightarrow \overline{\mathbb{R}}} \text{AUC}_{\sigma}(A).$$

The above proposition is a folklore result. However, we provide a proof for completeness, and the proof steps are also used to prove Observation 3.

Proof of Proposition 16 and Observation 3. To prove the above proposition, it suffices to show that for any scoring function σ , its achieved AUC is no greater than that achieved by using the log-likelihood ratio as the scoring function. In other words, we aim to prove that for any $\sigma : V_A \rightarrow \overline{\mathbb{R}}$,

$$\text{AUC}_{\mathcal{L}}(A) \geq \text{AUC}_{\sigma}(A).$$

Recall Definition 1. The AUC given a scoring function σ can be described using i.i.d. random variables $(X_U^+, C^+), (X_U^-, C^-) \sim \mathcal{D}$. We can express this quantity using indicator functions as follows

$$\text{AUC}_{\sigma}(A) = \mathbb{E}[\mathbf{1}\{\sigma(X_A^+) > \sigma(X_A^-)\} + \frac{1}{2}\mathbf{1}\{\sigma(X_A^+) = \sigma(X_A^-)\}|C^+ = 1, C^- = 0].$$

31:10 Feature Cross Search via Submodular Optimization

Note that $\mathbf{1}\{\sigma(X_A^+) > \sigma(X_A^-)\} + \mathbf{1}\{\sigma(X_A^+) = \sigma(X_A^-)\} + \mathbf{1}\{\sigma(X_A^+) < \sigma(X_A^-)\} = 1$, we have

$$\begin{aligned} \text{AUC}_\sigma(A) &= \frac{1}{2} + \mathbb{E} \left[\frac{1}{2} \mathbf{1}\{\sigma(X_A^+) > \sigma(X_A^-)\} - \frac{1}{2} \mathbf{1}\{\sigma(X_A^+) < \sigma(X_A^-)\} | C^+ = 1, C^- = 0 \right] \\ &= \frac{1}{2} + \frac{1}{2} \mathbb{E} [\mathbf{1}\{\sigma(X_A^+) > \sigma(X_A^-)\} - \mathbf{1}\{\sigma(X_A^+) < \sigma(X_A^-)\} | C^+ = 1, C^- = 0] \\ &= \frac{1}{2} + \frac{1}{2} \mathbb{E} [\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0]. \end{aligned} \quad (6)$$

To maximize the AUC, we focus on the term $\mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0]$. By symmetrizing this quantity, we obtain the following equations.

$$\begin{aligned} &\mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0] \\ &= \frac{1}{2} (\mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0] + \mathbb{E}[\text{sign}(\sigma(X_A^-) - \sigma(X_A^+)) | C^+ = 0, C^- = 1]) \\ &= \frac{1}{2} (\mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0] - \mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 0, C^- = 1]) \\ &= \frac{1}{2} \sum_{x_A^+, x_A^- \in V_A} (P_1[x_A^+]P_0[x_A^-] \text{sign}(\sigma(x_A^+) - \sigma(x_A^-)) - P_1[x_A^-]P_0[x_A^+] \text{sign}(\sigma(x_A^+) - \sigma(x_A^-))) \\ &= \frac{1}{2} \sum_{x_A^+, x_A^- \in V_A} (P_1[x_A^+]P_0[x_A^-] - P_1[x_A^-]P_0[x_A^+]) \text{sign}(\sigma(x_A^+) - \sigma(x_A^-)). \end{aligned} \quad (7)$$

The above expression can be upper bounded by the total variation distance between $P_1 \times P_0$ and $P_0 \times P_1$.

$$\begin{aligned} \mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0] &\leq \frac{1}{2} \sum_{x_A^+, x_A^- \in V_A} |P_1[x_A^+]P_0[x_A^-] - P_1[x_A^-]P_0[x_A^+]| \\ &= d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A). \end{aligned} \quad (8)$$

Note that whenever $P_1[x_A^+]P_0[x_A^-]$ or $P_1[x_A^-]P_0[x_A^+]$ is non-zero, the log-likelihood ratios $\mathcal{L}(x_A^+)$ and $\mathcal{L}(x_A^-)$, as well as $\mathcal{L}(x_A^+) - \mathcal{L}(x_A^-)$, are well defined on \mathbb{R} . Hence, if $P_1[x_A^+]P_0[x_A^-] - P_1[x_A^-]P_0[x_A^+] \neq 0$, one can show that

$$\text{sign}(P_1[x_A^+]P_0[x_A^-] - P_1[x_A^-]P_0[x_A^+]) = \text{sign}(\mathcal{L}(x_A^+) - \mathcal{L}(x_A^-)).$$

Consequently, all equality conditions in (8) can be achieved by using log-likelihood ratio as the scoring function, and we have

$$\begin{aligned} \mathbb{E}[\text{sign}(\sigma(X_A^+) - \sigma(X_A^-)) | C^+ = 1, C^- = 0] &\leq \mathbb{E}[\text{sign}(\mathcal{L}(X_A^+) - \mathcal{L}(X_A^-)) | C^+ = 1, C^- = 0] \\ &= d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A). \end{aligned} \quad (9)$$

Combining (6) and (9), we have the following bound that holds true for any σ ,

$$\text{AUC}_\sigma(A) \leq \text{AUC}_\mathcal{L}(A) = \frac{1}{2} + \frac{1}{2} d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A), \quad (10)$$

which completes the proof. \blacktriangleleft

According to Proposition 16 and equation (10), Observation 3 directly follows.

5 Total Variation of Commutator of Probability Measures

In this section, we prove Proposition 7, which states that the total variation of commutator of probability measures is a monotone submodular set function.

► **Proposition 7** (Proof in Section 5). *Let U be a finite index set. Assume that for every $a \in U$, there are a pair of probability measures P_0^a and P_1^a on a common sample space V_a . For any $A \subseteq U$, define the set function $F : 2^U \rightarrow \mathbb{R}_{\geq 0}$ by*

$$F(A) = d_{TV} \left(\bigotimes_{a \in A} P_1^a \times \bigotimes_{a \in A} P_0^a, \bigotimes_{a \in A} P_0^a \times \bigotimes_{a \in A} P_1^a \right). \quad (4)$$

The set function F is monotone and submodular.

5.1 Monotonicity Part of Proposition 7

We first show the monotonicity part.

Proof of Proposition 7 (Monotonicity). Let A and B be two subsets of U such that $A \subseteq B$. For any $A \subseteq U$, let $P_i^A = \bigotimes_{a \in A} P_i^a$. Using the above notation, we have $P_i^B = P_i^A \times P_i^{B \setminus A}$. By the definition of F , we have

$$\begin{aligned} F(A) &= d_{TV} (P_1^A \times P_0^A, P_0^A \times P_1^A) \\ &= \frac{1}{2} \sum_{x, y \in V_A} |P_1^A(x)P_0^A(y) - P_0^A(x)P_1^A(y)| \\ &= \frac{1}{2} \sum_{x, y \in V_A} \left| \sum_{z, w \in V_{B \setminus A}} P_1^A(x)P_0^A(y)P_1^{B \setminus A}(z)P_0^{B \setminus A}(w) \right. \\ &\quad \left. - \sum_{z, w \in V_{B \setminus A}} P_0^A(x)P_1^A(y)P_0^{B \setminus A}(z)P_1^{B \setminus A}(w) \right| \\ &\leq \frac{1}{2} \sum_{x, y \in V_A} \sum_{z, w \in V_{B \setminus A}} |P_1^A(x)P_0^A(y)P_1^{B \setminus A}(z)P_0^{B \setminus A}(w) - P_0^A(x)P_1^A(y)P_0^{B \setminus A}(z)P_1^{B \setminus A}(w)| \\ &= d_{TV}(P_1^A \times P_1^{B \setminus A} \times P_0^A \times P_0^{B \setminus A}, P_0^A \times P_0^{B \setminus A} \times P_1^A \times P_1^{B \setminus A}) \\ &= d_{TV}(P_1^B \times P_0^B, P_0^B \times P_1^B) \\ &= F(B). \end{aligned}$$

where the third equality is because

$$\sum_{z, w \in V_{B \setminus A}} P_1^{B \setminus A}(z)P_0^{B \setminus A}(w) = \sum_{z, w \in V_{B \setminus A}} P_0^{B \setminus A}(z)P_1^{B \setminus A}(w) = 1$$

and the inequality is a consequence of the triangle inequality. ◀

5.2 Submodularity Part of Proposition 7

To prove the submodularity part, we need the following lemmas.

► **Lemma 17** (General case, proof in the full version [5]). *Let $R, R' \in \Delta_{\Omega_1}$, $S, S' \in \Delta_{\Omega_2}$, and $P, Q \in \Delta_{\Omega}$, where $\Omega_1, \Omega_2, \Omega$ are finite sets. If $R \stackrel{f}{\sim} R'$ and $S \stackrel{g}{\sim} S'$, it holds that*

$$d_{TV}(R \times S \times P \times Q, R' \times S' \times Q \times P) - d_{TV}(R \times P \times Q, R' \times Q \times P) \\ - d_{TV}(S \times P \times Q, S' \times Q \times P) + d_{TV}(P \times Q, Q \times P) \leq 0.$$

We begin with the Bernoulli case where Ω_1 and Ω_2 in its statement are both $\{0, 1\}$ so that R, R', S, S' are all probability measures of a Bernoulli random variable.

► **Lemma 18** (Bernoulli case, proof in the full version [5]). *Let $R, R', S, S' \in \Delta_{\{0,1\}}$ such that $R \stackrel{f}{\sim} R'$ and $S \stackrel{f}{\sim} S'$, where f is a function on $\{0, 1\}$ such that $f(0) = 1$ and $f(1) = 0$. Let $P, Q \in \Delta_{\Omega}$, where Ω is a finite sample space. The following inequality holds*

$$d_{TV}(R \times S \times P \times Q, R' \times S' \times Q \times P) - d_{TV}(R \times P \times Q, R' \times Q \times P) \\ - d_{TV}(S \times P \times Q, S' \times Q \times P) + d_{TV}(P \times Q, Q \times P) \leq 0. \quad (11)$$

Proof sketch. To prove the Bernoulli case, we first show that under the summation (recall that according to Equation (1), the total variation distance is half of the L^1 distance, and the L^1 distance is the sum of the absolute value of the difference on each singleton), any term that involves an element of measure zero (with respect to P or Q) has no contribution to the expression on the left-hand side. We would like to emphasize that while the term itself may be non-zero, it will be canceled out under the summation. In our second step, we will consider quantities of the form $\sqrt{\frac{P(x)Q(y)}{Q(x)P(y)}}$ in which $Q(x)$ and $P(y)$ must be non-zero for all x and y . As a result, we have to eliminate elements of measure zero in first step by showing that their total contribution is zero.

As the second step, we perform a series of algebraic manipulations and substitutions and finally show that the opposite of left-hand side can be re-written as a quadratic $v^\top M v$, where v is a vector and M is a symmetric square matrix. Recall that the promised inequality claims that the left-hand side is non-positive (thus the opposite of the left-hand side is non-negative). Therefore, we will show it by establishing the positive semi-definiteness of M .

In fact, the matrix M is induced by a positive definite function. The problem of establishing the positive semi-definiteness of M reduces to the problem of proving that the function that induces M is positive definite. In light of the Bochner's theorem (see the full version [5]), we show its positive definiteness by computing its inverse Fourier transform, which turns out to be finite-valued and non-negative everywhere. ◀

The high-level strategy of proving Lemma 17 is to use Observation 19 to reduce the problem to the Bernoulli case (Lemma 18). The proof details can be found in the full version [5].

► **Observation 19.** *Let $P, P' \in \Delta_{\Omega}$ be such that $P \stackrel{f}{\sim} P'$ and $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}$ be a homogeneous bivariate function, i.e., $\phi(\lambda x, \lambda y) = \lambda \phi(x, y)$ holds for any $x, y, \lambda \in \mathbb{R}$. For every element $x \in \Omega$, we define the Bernoulli probability measure U_x on $\{0, 1\}$ such that $U_x(1) = \frac{P(x)}{P(x)+P'(x)}$ and $U'_x(1) = \frac{P'(x)}{P(x)+P'(x)}$. The following equation holds*

$$\sum_{x \in \Omega} \phi(P(x), P'(x)) = \sum_{x \in \Omega} \frac{P(x) + P'(x)}{2} (\phi(U_x(1), U'_x(1)) + \phi(U'_x(1), U_x(1))).$$

Before presenting the proof of Observation 19, we introduce the involutory swapping lemma, which is also used in the proof of Lemma 17. Intuitively, the involutory swapping lemma implies that two involution equivalent probability measures can be swapped inside a summation of a bivariate function.

► **Lemma 20** (Involutory swapping lemma). *Let $P, P' \in \Delta_\Omega$ be such that $P \stackrel{f}{\sim} P'$ and $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}$ be any bivariate function. Then we have*

$$\sum_{x \in \Omega} \phi(P(x), P'(x)) = \sum_{x \in \Omega} \phi(P'(x), P(x)).$$

Proof. Under the assumption of the lemma statement, we have

$$\begin{aligned} & \sum_{x \in \Omega} \phi(P(x), P'(x)) \\ &= \sum_{x \in \Omega} \phi(P'(f(x)), P(f(x))) \\ &= \sum_{x' \in \Omega} \phi(P'(x'), P(x')) \\ &= \sum_{x \in \Omega} \phi(P'(x), P(x)). \end{aligned}$$

The first equality is because for any $x \in \Omega$, we have $P(x) = P'(f(x))$ (by the definition of involution equivalence) and $P(f(x)) = P'(x)$ (Remark 10). The second equality is obtained by setting $x' = f(x)$ (this is because any involution map f is a bijection). The final equality is obtained by renaming x' to x . ◀

Proof of Observation 19. Under the assumption of the observation statement, we have

$$\begin{aligned} \sum_{x \in \Omega} \phi(P(x), P'(x)) &= \frac{1}{2} \sum_{x \in \Omega} \phi(P(x), P'(x)) + \frac{1}{2} \sum_{x \in \Omega} \phi(P(x), P'(x)) \\ &= \frac{1}{2} \sum_{x \in \Omega} \phi(P(x), P'(x)) + \frac{1}{2} \sum_{x \in \Omega} \phi(P'(x), P(x)) \\ &= \sum_{x \in \Omega} \frac{P(x) + P'(x)}{2} (\phi(U_x(1), U'_x(1)) + \phi(U'_x(1), U_x(1))). \end{aligned}$$

We use Lemma 20 in the second term on the second line and the third equality is because ϕ is homogeneous. ◀

We are in a position to show the submodularity part, which follows from Lemma 17.

Proof of Proposition 7 (Submodularity). To show that F is submodular, we need to check its definition that for any $A \subseteq U$ and $a, b \in U \setminus A$ such that $a \neq b$, it holds that

$$F(A \cup \{a\}) + F(A \cup \{b\}) \geq F(A \cup \{a, b\}) + F(A),$$

If we define $P_i^A = \times_{a \in A} P_i^a$, the above definition is equivalent to

$$\begin{aligned} & d_{TV}(P_1^a \times P_0^a \times P_1^A \times P_0^A, P_0^a \times P_1^a \times P_0^A \times P_1^A) \\ &+ d_{TV}(P_1^b \times P_0^b \times P_1^A \times P_0^A, P_0^b \times P_1^b \times P_0^A \times P_1^A) \\ &\geq d_{TV}(P_1^a \times P_0^a \times P_1^b \times P_0^b \times P_1^A \times P_0^A, P_0^a \times P_1^a \times P_0^b \times P_1^b \times P_0^A \times P_1^A) \\ &+ d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A). \end{aligned}$$

Re-arranging the terms yields

$$\begin{aligned}
& d_{TV}(P_1^a \times P_0^a \times P_1^b \times P_0^b \times P_1^A \times P_0^A, P_0^a \times P_1^a \times P_0^b \times P_1^b \times P_0^A \times P_1^A) \\
& - d_{TV}(P_1^a \times P_0^a \times P_1^A \times P_0^A, P_0^a \times P_1^a \times P_0^A \times P_1^A) \\
& - d_{TV}(P_1^b \times P_0^b \times P_1^A \times P_0^A, P_0^b \times P_1^b \times P_0^A \times P_1^A) \\
& + d_{TV}(P_1^A \times P_0^A, P_0^A \times P_1^A) \leq 0.
\end{aligned}$$

The above inequality follows from Lemma 17 if we set $P = P_1^A$, $Q = P_0^A$, $R = P_1^a \times P_0^a$, $R' = P_0^a \times P_1^a$, $S = P_1^b \times P_0^b$, and $S' = P_0^b \times P_1^b$. Note that $R \sim R'$ and $S \sim S'$ via the transpose involution (see Remark 11). \blacktriangleleft

6 Other Related Works

As discussed before, our problem falls in the category of feature engineering problems. Perhaps, the most studied problem in feature engineering is feature selection [10, 32, 33, 26, 11, 25, 18, 31]. In this problem, the goal is to select a small subset of the features to obtain a learning model with high accuracy and avoid over-fitting. Here we just mention a couple of feature selection algorithm related to submodular maximization and refer to [10] for an introduction to feature selection and many relevant references. [8] used the notion of weak submodularity to design and analyze feature selection algorithms. [16] used the submodularity of mutual information between the sensors to design a $(1 - 1/e)$ -approximation algorithm for sensor placements, which can be directly used for feature selection. However, as we show in Theorem 15, it is not possible to design such algorithms to maximize the mutual information between the features and the label.

Another related well-studied problem in this domain is vocabulary compression [2, 7, 28, 1]. The goal of vocabulary compression is to improve the learning and serving time, and in some cases to avoid overfitting. Vocabulary compression can be done by simple approaches such as filtering and naive bucketing, or more complex approaches such as mutual information maximization. [1] and [28] used clustering algorithms based on the Jensen-Shannon divergence to compress the vocabulary of features. [7] proposed an iterative algorithm that locally maximizes the mutual information between a feature and the label. Recently, [2] considered this problem for binary labels and presented a quasi-linear-time distributed approximation algorithm to maximize the mutual information between the feature and the label. There are polynomial-time local algorithms for binary labels that maximize the mutual information [17, 13], studied in the context of discrete memoryless channels.

[30] designed an integer programming based algorithm for feature cross search and applied it to learn generalized linear models using rule-based features. They show that this approach obtains better accuracy compared to that of the existing rule ensemble algorithms. [20] proposed a greedy algorithm for feature cross search and show that the greedy algorithm works well on a variety of datasets. Neither of these papers provide any theoretical guarantees for the performance of their algorithm.

7 Conclusion

In this paper, we considered the problem of feature cross search. We formulated it as a problem of maximizing the normalized area under the curve (AUC) of the linear model trained on the crossed feature column. We first established a hardness result that no algorithm can provide $n^{1/\log \log n}$ approximation for this problem unless the exponential time hypothesis

fails. Therefore, no polynomial algorithm can solve this problem unless $P = NP$. In light of its intractable nature, we motivated and assumed the naïve Bayes assumption. We related AUC to the total variation of the commutator of two probability measures. Under the naïve Bayes assumption, we demonstrated that the aforementioned total variation is monotone and submodular with respect to the set of selected feature columns to be crossed. As a result, a greedy algorithm can achieve a $(1 - 1/e)$ -approximation of the problem. Our proof techniques may be of independent interest. Finally, an empirical study showed that the greedy algorithm outperformed the baselines.

References

- 1 L Douglas Baker and Andrew Kachites McCallum. Distributional clustering of words for text classification. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 96–103. ACM, 1998.
- 2 Mohammadhossein Bateni, Lin Chen, Hossein Esfandiari, Thomas Fu, Vahab Mirrokni, and Afshin Rostamizadeh. Categorical feature compression via submodular optimization. In *International Conference on Machine Learning*, pages 515–523, 2019.
- 3 Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $o(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210. ACM, 2010.
- 4 Simon Byrne. A note on the use of empirical auc for evaluating probabilistic forecasts. *Electronic Journal of Statistics*, 10(1):380–393, 2016.
- 5 Lin Chen, Hossein Esfandiari, Gang Fu, Vahab S Mirrokni, and Qian Yu. Feature cross search via submodular optimization. *arXiv preprint arXiv:2107.02139*, 2021.
- 6 Yuxin Chen, S Hamed Hassani, Amin Karbasi, and Andreas Krause. Sequential information maximization: When is greedy near-optimal? In *Conference on Learning Theory*, pages 338–363, 2015.
- 7 Inderjit S Dhillon, Subramanyam Mallela, and Rahul Kumar. A divisive information-theoretic feature clustering algorithm for text classification. *Journal of machine learning research*, 3(Mar):1265–1287, 2003.
- 8 Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.
- 9 Susana Eyheramendy, David D Lewis, and David Madigan. On the naive bayes model for text categorization. In *9th International Workshop on Artificial Intelligence and Statistics*. Citeseer, 2003.
- 10 Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.
- 11 Nazrul Hoque, Dhruva K Bhattacharyya, and Jugal K Kalita. Mifs-nd: A mutual information-based feature selection method. *Expert Systems with Applications*, 41(14):6371–6385, 2014.
- 12 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- 13 Ken-ichi Iwata and Shin-ya Ozawa. Quantizer design for outputs of binary-input discrete memoryless channels using smawk algorithm. In *2014 IEEE International Symposium on Information Theory*, pages 191–195. IEEE, 2014.
- 14 Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014.
- 15 Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 324–331. AUAI Press, 2005.
- 16 Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of*

- the 5th international conference on Information processing in sensor networks*, pages 2–10. ACM, 2006.
- 17 Brian M Kurkoski and Hideki Yagi. Quantization of binary-input discrete memoryless channels. *IEEE Transactions on Information Theory*, 60(8):4544–4552, 2014.
 - 18 Nojun Kwak and Chong-Ho Choi. Input feature selection by mutual information based on parzen window. *IEEE transactions on pattern analysis and machine intelligence*, 24(12):1667–1671, 2002.
 - 19 Hui Lin. *Submodularity in natural language processing: algorithms and applications*. PhD thesis, University of Washington, 2012.
 - 20 Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.*, pages 1936–1945, 2019.
 - 21 Pasin Manurangsi. Almost-polynomial ratio hardness of approximating densest k-subgraph. In *STOC*, pages 954–961. ACM, 2017.
 - 22 Tom Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.
 - 23 Marko Mitrovic, Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Data summarization at scale: A two-stage submodular approach. In *ICML*, pages 3593–3602, 2018.
 - 24 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.
 - 25 Feiping Nie, Heng Huang, Xiao Cai, and Chris H Ding. Efficient and robust feature selection via joint $\ell_{2,1}$ -norms minimization. In *Advances in neural information processing systems*, pages 1813–1821, 2010.
 - 26 Monica Rogati and Yiming Yang. High-performing feature selection for text classification. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 659–661. ACM, 2002.
 - 27 Henry Scheffé. A useful convergence theorem for probability distributions. *The Annals of Mathematical Statistics*, 18(3):434–438, 1947.
 - 28 Noam Slonim and Naftali Tishby. The power of word clusters for text classification. In *23rd European Colloquium on Information Retrieval Research*, volume 1, page 200, 2001.
 - 29 Burak Turhan and Ayse Bener. Analysis of naive bayes’ assumptions on software fault data: An empirical study. *Data & Knowledge Engineering*, 68(2):278–290, 2009.
 - 30 Dennis Wei, Sanjeeb Dash, Tian Gao, and Oktay Günlük. Generalized linear rule models. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6687–6696, 2019.
 - 31 Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, pages 1954–1963. PMLR, 2015.
 - 32 Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. Feature selection for svms. In *Advances in neural information processing systems*, pages 668–674, 2001.
 - 33 Sepehr Abbasi Zadeh, Mehrdad Ghadiri, Vahab Mirrokni, and Morteza Zadimoghaddam. Scalable feature selection via distributed diversity maximization. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
 - 34 Yuanxing Zhang, Yichong Bai, Lin Chen, Kaigui Bian, and Xiaoming Li. Influence maximization in messenger-based social networks. In *GLOBECOM*, pages 1–6. IEEE, 2016.

An FPT Algorithm for the Embeddability of Graphs into Two-Dimensional Simplicial Complexes

Éric Colin de Verdière ✉

LIGM, CNRS, Univ Gustave Eiffel, F-77454 Marne-la-Vallée, France

Thomas Magnard ✉

LIGM, CNRS, Univ Gustave Eiffel, F-77454 Marne-la-Vallée, France

Abstract

We consider the embeddability problem of a graph G into a two-dimensional simplicial complex C : Given G and C , decide whether G admits a topological embedding into C . The problem is NP-hard, even in the restricted case where C is homeomorphic to a surface.

It is known that the problem admits an algorithm with running time $f(c)n^{O(c)}$, where n is the size of the graph G and c is the size of the two-dimensional complex C . In other words, that algorithm is polynomial when C is fixed, but the degree of the polynomial depends on C . We prove that the problem is fixed-parameter tractable in the size of the two-dimensional complex, by providing a deterministic $f(c)n^3$ -time algorithm. We also provide a randomized algorithm with expected running time $2^{c^{O(1)}}n^{O(1)}$.

Our approach is to reduce to the case where G has bounded branchwidth via an irrelevant vertex method, and to apply dynamic programming. We do not rely on any component of the existing linear-time algorithms for embedding graphs on a fixed surface; the only elaborated tool that we use is an algorithm to compute grid minors.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Graph algorithms; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Topology

Keywords and phrases computational topology, embedding, simplicial complex, graph, surface, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.32

Related Version *Full Version*: <https://arxiv.org/abs/2107.06236>

Funding Partially supported by the ANR projects Min-Max (ANR-19-CE40-0014) and SoS (ANR-17-CE40-0033).

Acknowledgements We would like to thank Arnaud de Mesmay for useful discussions.

1 Introduction

An embedding of a graph G into a host topological space X is a crossing-free topological drawing of G into X . The use and computation of graph embeddings is central in the communities of computational topology, topological graph theory, and graph drawing. A landmark result is the algorithm of Hopcroft and Tarjan [12], which allows to decide whether a given graph is planar (has an embedding into the plane) in linear time. Related results include more planarity testing algorithms [21], algorithms for embedding graphs on surfaces [18, 13] and for computing book embeddings [17], Hanani-Tutte theorems [24], and the theory of crossing numbers and planarization [2].

In this paper, we describe algorithms for deciding the embeddability of graphs into topological spaces that are, in a sense, as general as possible: two-dimensional simplicial complexes (or 2-complexes for brevity), which are made from vertices, edges, and triangles glued together. (We remark that every graph is embeddable in \mathbb{R}^3 , so considering higher-dimensional simplicial complexes is irrelevant.) In a previous article, jointly written with



© Éric Colin de Verdière and Thomas Magnard;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 32; pp. 32:1–32:17

Leibniz International Proceedings in Informatics



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

Mohar [6], we proved that, given a graph G and a 2-complex \mathcal{C} , one can decide whether G embeds into \mathcal{C} in polynomial time for fixed \mathcal{C} ; but the algorithm has running time $f(c) \cdot n^{O(c)}$, where n and c are the respective sizes of G and \mathcal{C} . Using a very different strategy, we describe algorithms for this problem, proving that it is fixed-parameter tractable in the complexity of the input complex:

► **Theorem 1.1.** *One can solve the embeddability problem of graphs into 2-dimensional simplicial complexes in deterministic $f(c)n^3$ time or in expected time $2^{c^{O(1)}}n^{O(1)}$, where c is the number of simplices of the input 2-complex, n is the number of vertices and edges of the input graph, and f is some computable function of c .*

2-complexes are much more general than surfaces, and tools that are suitable for studying embeddability of graphs on surfaces do not generalize. For example, the set of graphs embeddable on a given 2-complex is not closed under minor, which makes many tools for dealing with graphs on surfaces unsuitable for 2-complexes. Moreover, the complexity of some topological problems increase drastically when we consider 2-complexes instead of surfaces, e.g., deciding homeomorphism [20], or deciding the contractibility of curves [8, 16, 10]. Some other topological problems, such as the existence of a drawing a graph with at most k crossings in the plane or in a surface [14], can be recast as deciding whether the graph embeds on a certain 2-complex. For more detailed motivations, see [6, Introduction].

Comparison with previous works on surfaces

Since every surface is homeomorphic to a 2-complex, our problem has been largely considered in the special case where the input 2-complex is (homeomorphic to) a surface. That restricted problem is NP-hard [26], but several algorithms that are fixed-parameter tractable in the genus have been given, which we review now.

Mohar [18] has given an algorithm for this purpose that takes linear time in the input graph, for every fixed surface. This algorithm is very technical and relies on several other articles. The dependence on the genus is not made explicit, but seems to be doubly exponential [13].

Kawarabayashi et al., in an extended abstract [13], have given a simpler linear-time algorithm for this problem, but not all details are presented, which makes the approach hard to check [15, p. 3657, footnote].

General graph minor theory provides an algorithm for the same purpose. The graph minor theorem by Robertson and Seymour [23] implies that, for every fixed surface \mathcal{S} , there is a finite list of graphs $\mathcal{O}_{\mathcal{S}}$ such that a graph G can be embedded on \mathcal{S} if and only if G does not contain any graph in $\mathcal{O}_{\mathcal{S}}$ as a minor. Moreover, there is an algorithm that given any surface \mathcal{S} (specified by its genus and orientability) outputs the list $\mathcal{O}_{\mathcal{S}}$ [1], and there is an algorithm to decide whether a graph M is a minor of another graph G running, for fixed M , in time cubic in the size of G [22][7, Theorem 6.12]. These considerations thus lead to an algorithm to decide embeddability of a graph on a surface that runs, if the input surface is fixed, in cubic time in the size of the input graph.

Finally, in the same vein, Kociumaka and Pilipczuk [15] have studied the following more general problem than the embeddability problem of graphs on surfaces: Given a surface \mathcal{S} , a graph G , and an integer $k \geq 0$, is it possible to remove a set U of at most k vertices from G so that $G - U$ is embeddable on \mathcal{S} ? They provide an algorithm that is fixed-parameter tractable in k and the genus of \mathcal{S} , where the dependence on the genus is unspecified. In particular, as a special case, they decide the embeddability of a graph on a surface; however, they use one of the previous algorithms [18, 13] as a subroutine. The problem that we study, the embeddability of graphs on 2-complexes, is independent from the problem studied by Kociumaka and Pilipczuk, in the sense that there is, a priori, no obvious reduction from one problem to the other. However, we will reuse some ingredients from that paper.

Our algorithms, restricted to the case where we want to embed graphs on surfaces, are not as efficient as the existing algorithms mentioned above. Indeed, the deterministic one runs in cubic time in the size of the input graph (for a fixed complex); the dependence on the size of the complex is not made explicit, because the algorithm uses, as a subroutine, an algorithm to compute grid minors [22]. The second algorithm is randomized, because it uses an algorithmic version of the excluded grid theorem [3] that uses randomness; for every fixed surface, it runs in expected time that is a polynomial of fixed (but large) degree in the size of the input graph. However, our algorithms are independent from the existing algorithms for embedding graphs on surfaces; the only elaborated tool that we use is an algorithm to compute grid minors [3, 22].

Overview and structure of the paper

We use a standard strategy in graph algorithms and parameterized complexity (see, e.g., the book by Cygan et al. [7, Chapter 7]): we show by dynamic programming that the problem can be solved efficiently for graphs of bounded branchwidth, and then, using an irrelevant vertex method, we prove that one can assume without loss of generality that the input graph G has branchwidth bounded by a polynomial in the size of the input 2-complex. In the context of surface-embedded graphs, this paradigm has been used in the extended abstract by Kawarabayashi et al. [13] and in the article by Kociumaka and Pilipczuk [15]; our algorithm takes inspiration from the former, for the idea of the dynamic programming algorithm, and from the latter, for some arguments in the irrelevant vertex method. However, handling 2-complexes requires significantly more effort. More precisely, Theorem 1.1 follows immediately from the following two theorems.

► **Theorem 1.2** (algorithm for bounded branchwidth). *One can solve the embeddability problem of graphs into two-dimensional simplicial complexes in time $(c + w)^{O(c+w)}n$, where c is the number of simplices of the input 2-complex, and where n and w are the number of vertices and edges and the branchwidth of the input graph, respectively.*

► **Theorem 1.3** (algorithm to reduce branchwidth). *Let \mathcal{C} be a 2-complex with c simplices, and G a graph with n vertices and edges. We can correctly report that G is embeddable on \mathcal{C} , or correctly report that G is not embeddable on \mathcal{C} , or compute a subgraph H of G , of branchwidth polynomial in the number of simplices of \mathcal{C} , such that G embeds on \mathcal{C} if and only if H does:*

- *in deterministic time $f(c) \cdot n^3$ for some computable function f ,*
- *or in expected polynomial time.*

We now present the structure of the paper, indicating which techniques are used. We also emphasize which components would be simpler if we were just aiming for an algorithm for embedding graphs on surfaces.

We introduce some standard notions in Section 2.

Then, in Section 3, we show that we can make some simple assumptions on the input, and present data structures for representing 2-complexes and graphs embedded on them. If we restrict ourselves to the case where the input 2-complex is homeomorphic to a surface, we essentially consider combinatorial maps of graphs on surfaces, except that the graphs need not be cellularly embedded. The case of 2-complexes is largely more involved.

In Section 4, we show that if our input graph G has an embedding into our input 2-complex \mathcal{C} , then there exists an embedding of G on \mathcal{C} that is *sparse* with respect to a branch decomposition of G . This means that each subgraph of G induced by the leaves

of any subtree of the branch decomposition can be separated from the rest of G using a graph embedded on \mathcal{C} , called *partitioning graph*, of small complexity. We find that this new structural result, even in the surface case, is interesting and can prove useful in other contexts. If the target space were a surface, we could assume that G is 3-connected and has no loop or multiple edges, which would imply (still with some work) that *any* embedding of G would be sparse, but again the fact that we consider 2-complexes requires additional work.

In Section 5, we present the dynamic programming algorithm, which either determines the existence of an embedding of G on \mathcal{C} , or shows that no sparse embedding of G on \mathcal{C} exists (and thus no embedding at all, by the previous paragraph). The idea is to use bottom-up dynamic programming and to consider all regions of the 2-complex in which the subgraph of G (induced by a subtree of the branch decomposition) can be embedded. The complexity depends exponentially on the branchwidth of G .

The previous arguments, most notably in Section 4, implicitly assumed that, if G has an embedding into \mathcal{C} , it has a *proper and cellular embedding*, in particular, in which the faces are homeomorphic to disks. In Section 6, we show that we can assume this property. Essentially, we build all 2-complexes “smaller” than \mathcal{C} , such that G embeds on \mathcal{C} if and only if it embeds into (at least) one of these 2-complexes, and moreover if it is the case, it has an embedding into (at least) one of these 2-complexes that is proper and cellular. If \mathcal{C} were an orientable surface, we would just consider the surfaces of lower genus; but here a more sophisticated approach is needed.

The above ingredients allow to prove Theorem 1.2 (Section 7).

In Section 8, we show, using an irrelevant vertex method, that we can assume that G has branchwidth polynomial in the size of \mathcal{C} (Theorem 1.3).

2 Preliminaries

2.1 Graphs and branch decompositions

In this paper, graphs may have loops and multiple edges unless noted otherwise. Let G be a graph; as usual, we denote by $V(G)$ and $E(G)$ the sets of vertices and edges of G .

A **(rooted) branch decomposition** of G is a rooted tree \mathcal{B} in which:

- every node has degree either one (it is a **leaf**) or three (it is an **internal node**),
- the root is a leaf,
- each non-root leaf is labelled with an edge of G , and this labelling induces a bijection.

The vertices and edges of \mathcal{B} are called **nodes** and **arcs**, respectively. Each arc α of \mathcal{B} splits the tree \mathcal{B} into two subtrees \mathcal{B}_1 and \mathcal{B}_2 ; if, for $i = 1, 2$, we denote by E_i the set of labels appearing in T_i , we see that α naturally induces a partition (E_1, E_2) of the set of edges of G (if α is the arc incident to the root, then one part of the partition is empty). The **middle set** associated to α is the set of vertices of G which are the endpoints of at least one edge in E_1 and at least one edge in E_2 . The **width** of \mathcal{B} is the maximum size of a middle set associated to an arc of \mathcal{B} . The **branchwidth** of G is the minimum width of its (rooted) branch decompositions.

The usual definition of a branch decomposition is identical, except that the tree is unrooted, and thus the leaves are in bijection with the edges of G . Our definition turns out to be more convenient to use in the dynamic program. The difference is cosmetic, as one can transform one kind of branch decomposition into the other easily while preserving the width.

2.2 Surfaces

We will assume some familiarity with surface topology; see, e.g., [19, 25, 4] for suitable introductions under various viewpoints. We recall some basic definitions and properties. A **surface** (without boundary) \mathcal{S} is a compact, connected Hausdorff topological space in which every point has an open neighborhood homeomorphic to the open disk. Up to homeomorphism, every surface \mathcal{S} is obtained from a sphere by:

- either removing $g/2$ open disks and attaching a handle (a torus with an open disk removed) to each resulting boundary component, where g is an even, nonnegative integer called the (*Euler*) **genus** of \mathcal{S} ; in this case, \mathcal{S} is **orientable**;
- or removing g open disks and attaching a Möbius band to each resulting boundary component, for a positive number g called the (non-orientable) **genus** of \mathcal{S} ; in this case, \mathcal{S} is **non-orientable**.

A **possibly disconnected surface** is a disjoint union of surfaces.

A **surface with boundary** is obtained from a surface (without boundary) by removing a finite set of interiors of disjoint closed disks. The boundary of each disk forms a **boundary component** of \mathcal{S} . The **genus** of \mathcal{S} is defined as the genus of the original surface without boundary. Equivalently, a surface with boundary is a compact, connected Hausdorff topological space in which every point has an open neighborhood homeomorphic to the open disk or the closed half disk $\{(x, y) \in \mathbb{R}^2 \mid y \geq 0, x^2 + y^2 < 1\}$.

2.3 2-complexes

A **2-complex** (or two-dimensional simplicial complex) is an abstract simplicial complex of dimension at most two: a finite set of 0-simplices called **vertices**, 1-simplices called **edges**, and 2-simplices called **triangles**. Each edge is a pair of vertices, and each triangle is a triple of vertices; moreover, each subset of size two in a triangle must be an edge.

Each 2-complex \mathcal{C} corresponds naturally to a topological space, obtained as follows: Start with one point per vertex in \mathcal{C} ; connect them by segments as indicated by the edges in \mathcal{C} ; similarly, for every triangle in \mathcal{C} , create a triangle whose boundary is made of the three edges contained in that triangle. By abuse of language, we identify \mathcal{C} with that topological space.

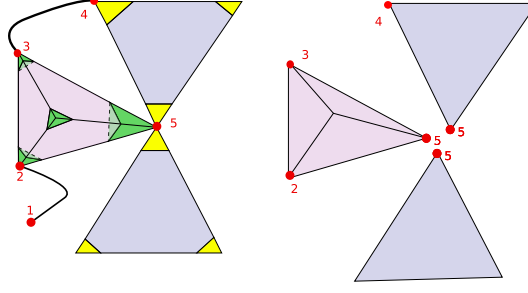
2.4 Graph embeddings

Each graph has a natural associated topological space (for graphs without loops or multiple edges, this is a specialization of the definition for 2-complexes). An **embedding** Γ of a graph G into a 2-complex \mathcal{C} is an injective continuous map from (the topological space associated to) G to (the topological space associated to) \mathcal{C} . A **face** of Γ is a connected component of the complement of the image of Γ in \mathcal{C} .

3 2-complexes and their data structures

3.1 Some preprocessing

A **3-book** is a topological space obtained from three triangles by considering one side per triangle and identifying these three sides together into a single edge. We say that a 2-complex \mathcal{C} **contains a 3-book** if \mathcal{C} contains three distinct triangles that share a common edge. Because every graph can be embedded into a 3-book [6, Proposition 3.1], we have the following proposition:



■ **Figure 1** On the left: A 2-complex with 5 singular points, numbered from 1 to 5, and 2 isolated edges (one between 3 and 4 and one between 1 and 2) where, at singular points, the cones are in green and the corners in yellow. On the right: the corresponding detached surface.

► **Proposition 3.1.** *To decide the embeddability of a graph G on a 2-complex \mathcal{C} , we can without loss of generality, after a linear-time preprocessing, assume the following properties on the input:*

- \mathcal{C} has no 3-book and no connected component that is reduced to a single vertex;
- G has no connected component reduced to a single vertex, and at most one connected component homeomorphic to a segment.

In the rest of this article, without loss of generality, we implicitly assume that \mathcal{C} and G satisfy the properties stated in Proposition 3.1.

3.2 Structure of 2-complexes without 3-book or isolated vertex

Let \mathcal{C} be a 2-complex without 3-book or isolated vertex, and let p be a vertex of \mathcal{C} . Following [6, Section 2.2], we describe the possible neighborhoods of p in \mathcal{C} . A **cone at p** is a cyclic sequence of triangles t_1, \dots, t_k, t_1 ($k \geq 3$), all incident to p , such that, for each $i = 1, \dots, k$, the triangles t_i and t_{i+1} (where $t_{k+1} = t_1$) share an edge incident with p , and any other pair of triangles have only p in common. A **corner at p** is a sequence of distinct triangles t_1, \dots, t_k , all incident to p , such that, for each $i = 1, \dots, k-1$, the triangles t_i and t_{i+1} share an edge incident with p , any other pair of these triangles have only p in common, and no other triangle in \mathcal{C} shares an edge incident with p and belonging to one of t_1, \dots, t_k . An **isolated segment at p** is an edge incident to p but not incident to any triangle. The cones, corners, and isolated segments at p form the **link components at p** .

The set of edges and triangles incident with a given vertex p of \mathcal{C} are uniquely partitioned into cones, corners, and isolated segments. We say that p is a **regular point** if all the edges and triangles incident to p form a single cone or corner; in that case, p has an open neighborhood homeomorphic to a disk or a closed half-disk. Otherwise, p is a **singular point**. See Figure 1, left, for an illustration.

Detaching a singular point p in \mathcal{C} consists of the following operation: replace p with new vertices, one for each cone, corner, and isolated segment at p . Detaching all singular points of a 2-complex (without 3-book) yields a disjoint union of (1) isolated segments and (2) a surface, possibly disconnected, possibly with boundary, called the **detached surface** (see Figure 1, right). The trace of the singular points on the detached surface are the **marked points**. Conversely, \mathcal{C} can be obtained from a surface (possibly disconnected, possibly with boundary) and a finite set of segments by choosing finitely many subsets of points and identifying the points in each subset together.

The **boundary** of \mathcal{C} is the closure of the set of points of \mathcal{C} that have an open neighborhood homeomorphic to a closed half-plane. Equivalently, it is the union of the edges of \mathcal{C} incident with a single triangle.

3.3 Topological data structure for 2-complexes

Any 2-complex \mathcal{C} without 3-book or isolated vertex is obtained from the detached surface and a set of disjoint segments, by identifying finitely many finite subsets of points. It is thus easy to describe a **topological data structure** for storing 2-complexes, by storing the topology of the detached surface and the segments, and recording additionally the identification of points to obtain the complex. The **size** of \mathcal{C} is the sum of the number of isolated segments, the number of connected components of the detached surface, the total genus of the detached surface, the total number of boundary components of the detached surface, and the total number of marked points of the detached surface (the occurrences of the singular points). This is, up to a constant factor, the size of the topological data structure indicated above, if the genus is stored in unary. Any 2-complex described in the usual combinatorial way can be converted in polynomial time into this representation. Thus, **in the rest of this article, without loss of generality, we implicitly assume that \mathcal{C} is given in the form of the above topological data structure.**

Moreover, given two 2-complexes in the form above, deciding whether they are homeomorphic essentially amounts to testing whether the corresponding data structures are isomorphic, which leads to the following lemma (the running time might be improvable, but this suffices for our purposes):

► **Lemma 3.2.** *Given two 2-complexes \mathcal{C} and \mathcal{C}' , given in the topological representation above, of sizes c and c' respectively, we can decide whether \mathcal{C} and \mathcal{C}' are homeomorphic in time $(c + c')^{O(c+c')}$.*

3.4 Proper and cellular graph embeddings on 2-complexes

Let \mathcal{C} be a 2-complex with size c , G a graph, and Γ an embedding of G on \mathcal{C} . The embedding Γ is **proper** if:

- the image of Γ meets the boundary of \mathcal{C} only on singular points;
- the vertices of Γ cover the singular points of \mathcal{C} .

The embedding Γ is **cellular** if each face of Γ is an open disk plus possibly some part of the boundary of \mathcal{C} . We emphasize that this definition slightly departs from the standard one. Moreover, we will only consider cellular embeddings that are proper.

We will use a data structure to store possibly non-cellular embeddings of graphs on surfaces [5, Section 2.2]. Such a data structure is based on the gem representation of cellular graph embeddings [9, Section 2], but store additional information about the topology of the faces. It is important to remark that this data structure also allows to recover the topology of the underlying surface.

Let Γ be a proper graph embedding of a graph G on a 2-complex \mathcal{C} (under the assumptions of Proposition 3.1). Let \mathcal{S} be the detached surface of \mathcal{C} . Because Γ is proper, it naturally induces an embedding Γ' , of another graph G' , on \mathcal{S} ; some vertices of G located on singular points of \mathcal{C} are duplicated in G' , the vertices of G located in the relative interior of isolated segments are absent from G' , and the edges of G not in G' are edges on the isolated segments of \mathcal{C} . Our data structure, called **combinatorial map**, for storing the graph embedding Γ and the 2-complex \mathcal{C} consists of storing (1) the graph embedding Γ' on \mathcal{S} , as indicated in the previous paragraph, (2) the isolated segments of \mathcal{C} , together with, for each such isolated segment, an ordered list alternating vertices and edges of Γ (or, instead of an edge, a mark indicating the absence of such an edge in the region of the isolated segment between the incident vertices), (3) the identifications of vertices of Γ' that are needed to recover Γ (and thus implicitly \mathcal{C}).

Isomorphisms between combinatorial maps are defined in the obvious way, similar to the concept of isomorphism between topological data structures: Two combinatorial maps are isomorphic if there is an isomorphism between the combinatorial maps restricted to the detached surfaces, isomorphisms between the maps on each isolated segments, and such that incidences are preserved on the singular points. We can easily test isomorphism between two combinatorial maps of size k and k' , respectively, in $(k + k')^{O(k+k')}$ time.

We will need an algorithm to enumerate all proper embeddings of small graphs on a given 2-complex. This is achieved by a brute-force algorithm:

► **Lemma 3.3.** *Let \mathcal{C} be a 2-complex of size c and k an integer. We can enumerate the $(c + k)^{O(c+k)}$ combinatorial maps of graphs with at most k vertices and at most k edges properly embedded on \mathcal{C} in $(c + k)^{O(c+k)}$ time.*

4 Partitioning graphs

Let \mathcal{C} be a 2-complex and G a graph, which satisfy the properties of Proposition 3.1. In this section, we lay the structural foundations of the dynamic programming algorithm, described in the next section (Proposition 5.1). The goal, in this section and the following one, is to obtain an algorithm that takes as input \mathcal{C} and G , and, in time FPT in the size of \mathcal{C} and the branchwidth of G , reports correctly one of the following two statements:

- G has no proper cellular embedding on \mathcal{C} ,
- G has an embedding on \mathcal{C} .

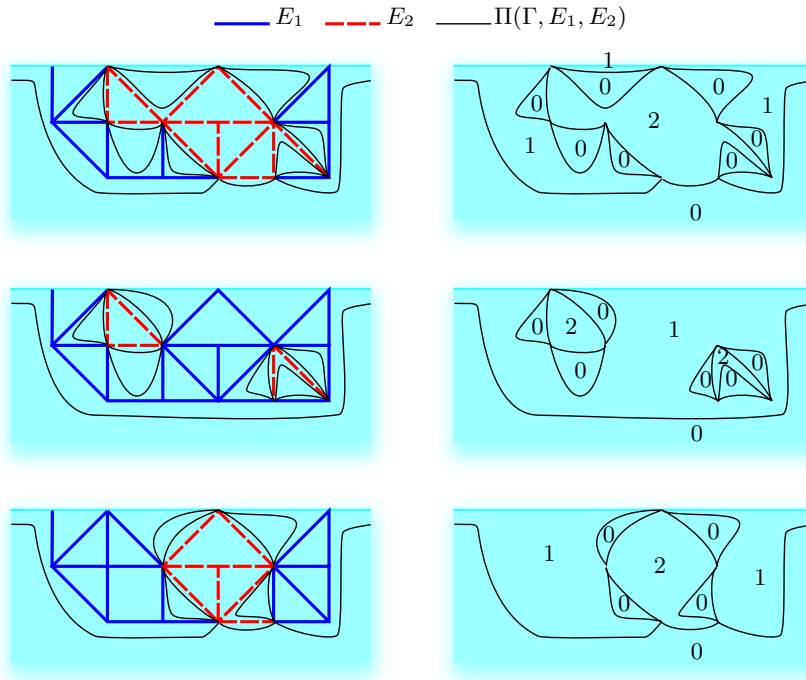
This algorithm uses dynamic programming on a rooted branch decomposition of G . When processing a node of the rooted branch decomposition, it considers embeddings of the subgraph of G induced by the edges in the leaves of the subtree rooted at that node in a region of \mathcal{C} . This region will be delimited by a *partitioning graph* embedded on \mathcal{C} . Our dynamic program will roughly guess the partitioning graph at each node of the rooted branch decomposition. For this purpose, we need that, if G has a proper cellular embedding on \mathcal{C} , it has such an embedding that is *sparse*: at each node of the rooted branch decomposition of G , the partitioning graph corresponding to the embedding of the induced subgraph is small (its size is upper-bounded by a function of the branchwidth of G and of the size of \mathcal{C}). The goal of this section is to prove that this is indeed the case.

Let (E_1, \dots, E_k) be an (ordered) partition of the edge set $E(G)$ of G . (We will only use the cases $k = 2$ or $k = 3$.) The **middle set** of (E_1, \dots, E_k) is the set of vertices of G whose incident edges belong to at least two sets E_i .

Let Γ be a proper cellular embedding of G on \mathcal{C} . Since Γ is cellular, every boundary of \mathcal{C} is incident to at least one vertex of Γ . Let $\hat{\Gamma}$ be obtained from Γ by adding edges as follows: for any pair of vertices u and v of Γ consecutive along a given boundary component of \mathcal{C} , we connect u and v via a new edge that runs along the boundary component. For each (ordered) partition (E_1, \dots, E_k) of the edge set of G , we let \hat{E}_1 be the union of E_1 and of these new edges, and $\hat{E}_i = E_i$ for each $i \neq 1$; thus, $(\hat{E}_1, \dots, \hat{E}_k)$ is a partition of the set of edges of $\hat{\Gamma}$.

The **partitioning graph** $\Pi(\Gamma, E_1, \dots, E_k)$ (or more concisely Π) associated to Γ and (E_1, \dots, E_k) is a graph properly embedded on \mathcal{C} (but possibly non-cellularly), with labels on its faces, defined as follows:

- The vertex set of Π is the union of the singular points of \mathcal{C} and of (the images under Γ of) the middle set of E_1, \dots, E_k .
- The relative interiors of the edges of Π are disjoint from the edges of $\hat{\Gamma}$ and from the isolated segments of \mathcal{C} . Let f be a face of $\hat{\Gamma}$ (which is homeomorphic to an open disk plus possibly some points of the boundary of \mathcal{C}). Let us describe the edges of Π inside f .



■ **Figure 2** Construction of the partitioning graph $\Pi = \Pi(\Gamma, E_1, E_2)$, for three choices of the partition (E_1, E_2) of the same embedding Γ . Only a part of the 2-complex \mathcal{C} is shown, with a boundary at the upper part, and without singular point. Left: The graph embeddings Γ (in thick lines) and Π (in thin lines). Right: The sole graph Π , together with the labelling of its faces.

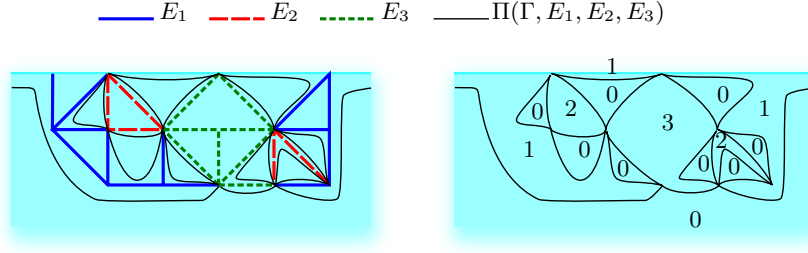
If, for some $i \in \{1, \dots, k\}$, the boundary of f is comprised only of edges of $\hat{\Gamma}$ that lie in a single set \hat{E}_i , then Π contains no edge inside f . Otherwise, the boundary of f is a succession of edges of $\hat{E}_1, \hat{E}_2, \dots, \hat{E}_k$. The edges of Π inside f run along the boundary of f ; for each $i \in \{1, \dots, k\}$, for each (maximal) group of consecutive edges in \hat{E}_i along the boundary of f , we create an edge of Π that runs along this group, with endpoints the corresponding vertices on the boundary of f (see Figures 2 and 3). These vertices either are in the middle set of (E_1, \dots, E_k) , or lie on the boundary of \mathcal{C} (and thus on singular points of \mathcal{C}).

It follows from the construction that $\hat{\Gamma}$ and Π intersect only at common vertices.

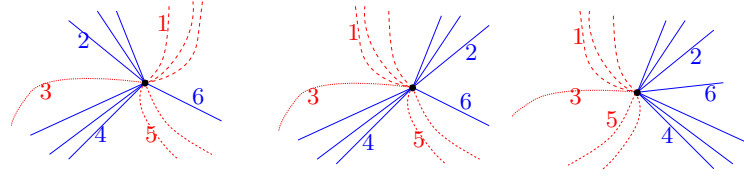
- Each face of Π is labelled by an integer in $\{0, \dots, k\}$ as follows: faces of Π containing edges in \hat{E}_i are labelled i , and the other ones are labelled 0. By construction of the graph Π , each face of Π contains edges from at most one set \hat{E}_i , so this labelling is well defined.

Henceforth, we fix a rooted branch decomposition \mathcal{B} of G , the root of which is denoted by ρ . Every arc α of \mathcal{B} naturally partitions $E(G)$ into two parts E_1 and E_2 , in which E_1 is the part on the same side as ρ ; this (ordered) partition is the **edge partition associated to α** . Recall that Γ is a proper and cellular embedding of G on \mathcal{C} ; we let $\Pi(\Gamma, \alpha)$ be $\Pi(\Gamma, E_1, E_2)$. Similarly, every node ν of \mathcal{B} naturally partitions $E(G)$ into three parts E_1, E_2 , and E_3 , in which E_1 is the part on the same side as ρ ; this partition is the **edge partition associated to ν** ; we let $\Pi(\Gamma, \nu)$ be $\Pi(\Gamma, E_1, E_2, E_3)$.

We say that Γ is **sparse** (with respect to \mathcal{B}) if the following conditions hold, letting c be the size of \mathcal{C} and w the width of \mathcal{B} :



■ **Figure 3** The partitioning graph $\Pi = \Pi(\Gamma, E_1, E_2, E_3)$. Left: The graph embeddings Γ (in thick lines) and Π (in thin lines). Right: The sole graph Π , together with the labelling of its faces.



■ **Figure 4** Left: A vertex with 6 intervals, numbered from 1 to 6. Middle: The cyclic order obtained by applying the first type of simplification operation on intervals 1 and 2. After the simplification, the intervals 1 and 3 are merged into a single one, and similarly for the intervals 2 and 6. Right: The cyclic order obtained by applying the second type of simplification to the configuration on the left, on pairs of intervals $\{1, 2\}$ and $\{4, 5\}$. After the simplification, the intervals 1, 3, and 5 are merged, and similarly for the intervals 2, 6, and 4.

- For each arc α of \mathcal{B} , the graph $\Pi(\Gamma, \alpha)$ has at most $74c + 26w$ edges;
- similarly, for each internal node ν of \mathcal{B} , the graph $\Pi(\Gamma, \nu)$ has at most $3(74c + 26w)$ edges.

The result of this section is the following.

► **Proposition 4.1.** *Let \mathcal{C} be a 2-complex and G a graph, which satisfy the properties of Proposition 3.1. Let \mathcal{B} be a rooted branch decomposition of G . Assume that G has a proper cellular embedding on \mathcal{C} . Then it has a proper cellular embedding Γ on \mathcal{C} that is sparse (with respect to \mathcal{B}).*

4.1 Monogons and bigons

A **monogon** of a graph Π embedded on a 2-complex \mathcal{C} is a face of Π that is an open disk whose boundary is a single edge of Π (a loop). Similarly, a **bigon** of Π is a face of Π that is an open disk whose boundary is the concatenation of two edges of Π (possibly the same edge appearing twice). The following general lemma on graphs embedded on surfaces without monogons or bigons will be used.

► **Lemma 4.2.** *Let \mathcal{S} be a surface of genus g without boundary. Let Π be a graph embedded on \mathcal{S} , not necessarily cellularly. Assume that Π has no monogon or bigon. Then $|E(\Pi)| \leq \max\{0, 3g + 3|V(\Pi)| - 6\}$.*

4.2 Vertex simplifications

The proof of Proposition 4.1 starts with any proper cellular embedding of Γ , and iteratively changes the cyclic ordering of edges around vertices in a specific way. Let (E_1, E_2) be an (ordered) partition of $E(G)$, let v be a vertex of G , and let C be a link component at v (if

the image of v under Γ is a singular point, there may be several such link components). We restrict our attention to the edges of $\hat{\Gamma}$ incident to v and belonging to C , in cyclic order around v . For $i = 1, 2$, an **interval** (at v , relatively to (\hat{E}_1, \hat{E}_2)) is a maximal contiguous subsequence of edges in this cyclic ordering that all belong to \hat{E}_i ; the interval is labelled i . **Simplifying** v (with respect to (E_1, E_2)) means changing the cyclic ordering of the edges of $\hat{\Gamma}$ incident to v in C by one of the two following operations (Figure 4):

1. either exchanging two consecutive intervals in that ordering, in a way that the ordering of the edges in each interval is preserved; this operation is allowed only if v is incident to at least four intervals;
2. or performing the previous operation twice, on two disjoint pairs of consecutive intervals in that ordering; this is allowed only if v is incident to at least six intervals.

We will rely on the following lemma.

► **Lemma 4.3.** *Let Γ be a proper cellular embedding of G on \mathcal{C} , and let (E_1, E_2) be an (ordered) partition of $E(G)$. Let Γ' be another proper cellular embedding of G , obtained from Γ by simplifying one or two vertices with respect to (E_1, E_2) , while keeping the other cyclic orderings unchanged. Then:*

1. $|E(\Pi(\Gamma', E_1, E_2))| < |E(\Pi(\Gamma, E_1, E_2))|$;
2. for each (ordered) partition $(\tilde{E}_1, \tilde{E}_2)$ of $E(G)$ such that $\hat{\tilde{E}}_i \subseteq \hat{E}_j$ for some $i, j \in \{1, 2\}$, we have $|E(\Pi(\Gamma', \tilde{E}_1, \tilde{E}_2))| \leq |E(\Pi(\Gamma, \tilde{E}_1, \tilde{E}_2))|$.

Proof. The proof is based on the following easy but key observations (the second one will be reused later):

- A simplification of v strictly decreases the number of intervals at v ;
- the number of half-edges of $\Pi(\Gamma, E_1, E_2)$ at v in the link component C equals twice the number of intervals associated to (\hat{E}_1, \hat{E}_2) at v in C .

The first point of the lemma immediately follows. For the second point, let us consider, in the cyclic ordering around v in C , a maximal contiguous sequence of edges in $\hat{\tilde{E}}_i$. Since $\hat{\tilde{E}}_i \subseteq \hat{E}_j$, when simplifying with respect to (E_1, E_2) , this sequence is still contiguous in the new embedding Γ' . It follows that the number of intervals associated to $(\hat{\tilde{E}}_1, \hat{\tilde{E}}_2)$ does not increase when replacing Γ with Γ' . ◀

4.3 Rearranging Γ with respect to an edge partition

We can now prove the following lemma:

► **Lemma 4.4.** *Let Γ be a proper cellular embedding of G on \mathcal{C} , and let (E_1, E_2) be an (ordered) partition of $E(G)$. There exists a proper cellular embedding Γ' of G such that:*

- $|E(\Pi(\Gamma', E_1, E_2))| \leq 74c + 26w$, where w is the size of the middle set of (E_1, E_2) ;
- for each (ordered) partition $(\tilde{E}_1, \tilde{E}_2)$ of $E(G)$ such that $\hat{\tilde{E}}_i \subseteq \hat{E}_j$ for some $i, j \in \{1, 2\}$, we have $|E(\Pi(\Gamma', \tilde{E}_1, \tilde{E}_2))| \leq |E(\Pi(\Gamma, \tilde{E}_1, \tilde{E}_2))|$.

Sketch of proof. Let $\Pi := \Pi(\Gamma, E_1, E_2)$. We assume that Π has “many monogons or bigons” and show that there is another cellular embedding Γ' of G such that:

- $|E(\Pi(\Gamma', E_1, E_2))| < |E(\Pi(\Gamma, E_1, E_2))|$;
- for each (ordered) partition $(\tilde{E}_1, \tilde{E}_2)$ of $E(G)$ such that $\hat{\tilde{E}}_i \subseteq \hat{E}_j$ for some $i, j \in \{1, 2\}$, we have $|E(\Pi(\Gamma', \tilde{E}_1, \tilde{E}_2))| \leq |E(\Pi(\Gamma, \tilde{E}_1, \tilde{E}_2))|$.

Intuitively, if Π has many monogons attached to a single vertex, or many bigons glued together, we can change the embedding Γ in a way that leads to vertex simplifications. By repeatedly iterating this argument, and up to replacing Γ with Γ' , this implies that we can assume without loss of generality that Π has “not too many monogons or bigons”. Lemma 4.2 then implies that Π has at most $74c + 26w$ edges, which concludes. ◀

4.4 Proof of Proposition 4.1

Proof of Proposition 4.1. Let \mathcal{B} be a rooted branch decomposition of G , and let Γ be a proper cellular embedding of G on \mathcal{C} . We consider each arc α of the rooted branch decomposition in turn, in an arbitrary order. For each such arc, we modify Γ by applying Lemma 4.4. We first claim that after these iterations, for each arc α of \mathcal{B} , we have $|E(\Pi(\Gamma, \alpha))| \leq 74c + 26w$.

Immediately after applying the above procedure to an arc $\tilde{\alpha}$ of \mathcal{B} , corresponding to the (ordered) partition $(\tilde{E}_1, \tilde{E}_2)$ of $E(G)$, we have $|E(\Pi(\Gamma, \tilde{E}_1, \tilde{E}_2))| \leq 74c + 26w$. We now prove that subsequent applications of Lemma 4.4 to other arcs of the rooted branch decomposition do not increase this number of edges. Indeed, let α be another arc, corresponding to the (ordered) partition (E_1, E_2) of $E(G)$, to which we apply Lemma 4.4. The arc α partitions the nodes of the tree \mathcal{B} into two sets N_1 and N_2 , and similarly $\tilde{\alpha}$ partitions the nodes of the tree \mathcal{B} into two sets \tilde{N}_1 and \tilde{N}_2 . Because \mathcal{B} is a tree, we have $\tilde{N}_i \subseteq N_j$ for some $i, j \in \{1, 2\}$. This implies that $\hat{\tilde{E}}_i \subseteq \hat{E}_j$ for some $i, j \in \{1, 2\}$; so the second item of Lemma 4.4 implies that the number of edges of $\Pi(\Gamma, \tilde{E}_1, \tilde{E}_2)$ does not increase when processing arc α . This proves the claim.

Finally, there remains to prove that, for each internal node ν of \mathcal{B} , the graph $\Pi(\Gamma, \nu)$ has at most $3(74c + 26w)$ edges. This follows relatively easily from the above claim. \blacktriangleleft

5 Dynamic programming algorithm

The result of this section is the following proposition.

► **Proposition 5.1.** *Let \mathcal{C} be a 2-complex and G a graph, which satisfy the properties of Proposition 3.1. Let c be the size of \mathcal{C} and n the number of vertices and edges of G . Let \mathcal{B} be a rooted branch decomposition of G of width w . In $(c + w)^{O(c+w)}n$ time, one can report one of the following statements, which is true:*

- G has no sparse proper cellular embedding into \mathcal{C} ;
- G has an embedding into \mathcal{C} .

(Proposition 4.1 implies that we can remove the adjective “sparse” in the above proposition.)

5.1 Bounding graphs

Let \mathcal{B} be a rooted branch decomposition of G of width w . Recall (see Section 2.1) that the root ρ of \mathcal{B} is a leaf associated to no edge of G . Our algorithm will use dynamic programming in the rooted branch decomposition. For each arc α of \mathcal{B} , let G_α be the subgraph of G induced by the edges of G corresponding to the leaves of the subtree of \mathcal{B} rooted at α . The general idea is that we compute all possible relevant embeddings of G_α in subregions of \mathcal{C} . Such subregions will be delimited by a graph embedded on \mathcal{C} of small complexity. For the dynamic program to work, we also need to keep track of the location of the vertices in the middle set of α . More precisely, a **bounding graph** for G_α is a proper labelled graph embedding Π on \mathcal{C} (but possibly non-cellular), such that:

- some vertices of Π are labelled; these labels are exactly the vertices of the middle set associated with α , and each label appears exactly once;
- each unlabelled vertex of Π is mapped to a singular point of \mathcal{C} ;
- each face of Π is labelled 0, 1, or 2;

- G_α has an embedding Γ_α that **respects** Π : each vertex of Π labelled v is mapped, under Π , to the image of v in Γ_α ; moreover, the relative interior of each edge of Γ_α lies in the interior of a face of Π labelled 2.

A bounding graph for G_α is **sparse** if it has at most $74c + 26w$ edges. Remark that, if Γ is a sparse proper cellular embedding of G on \mathcal{C} (as defined in Section 4), then $\Pi(\Gamma, \alpha)$ is a sparse bounding graph for the restriction of Γ to G_α .

Henceforth, we regard two (labelled) properly embedded graphs as equal if and only if their (labelled) combinatorial maps are isomorphic. A list \mathcal{L}_α of sparse bounding graphs for G_α is **exhaustive** if the following condition holds: If G has a sparse proper cellular embedding on \mathcal{C} , then for each such embedding Γ , the (combinatorial map of the) graph $\Pi(\Gamma, \alpha)$ is in \mathcal{L}_α .

The induction step for the dynamic programming algorithm is the following.

► **Proposition 5.2.** *Let ν be a non-root node of \mathcal{B} and α be the arc of \mathcal{B} incident to ν that is the closest to the root ρ . Assume that, for each arc $\beta \neq \alpha$ of \mathcal{B} incident to ν , we are given an exhaustive list of sparse bounding graphs for G_β . Then we can, in $(c + w)^{O(c+w)}$ time, compute an exhaustive list of $(c + w)^{O(c+w)}$ sparse bounding graphs for G_α .*

Assuming Proposition 5.2, the proof of which is deferred to the next subsection, it is easy to prove Proposition 5.1:

Proof of Proposition 5.1, assuming Proposition 5.2. We apply the algorithm of Proposition 5.2 in a bottom-up manner in the rooted branch decomposition \mathcal{B} . Let α be the arc of \mathcal{B} incident with the root node ρ . We end up with an exhaustive list of sparse bounding graphs for $G_\alpha = G$. By definition of a bounding graph, if this list is non-empty, then G has an embedding on \mathcal{C} . On the other hand, by definition of an exhaustive list, if this list is empty, then G has no sparse proper cellular embedding on \mathcal{C} .

There are $O(n)$ recursive calls, each of which takes $(c + w)^{O(c+w)}$ time. ◀

5.2 The induction step: Proof of Proposition 5.2

Proof of Proposition 5.2. First case. Let us first assume that ν is a (non-root) leaf of \mathcal{B} ; thus, G_α is a single edge uv . Using Lemma 3.3, we compute *all* the labelled combinatorial maps of sparse bounding graphs for G_α . This is clearly an exhaustive list. Indeed, assume that G has a sparse proper cellular embedding Γ on \mathcal{C} ; by sparsity, $\Pi(\Gamma, \alpha)$ has at most $74c + 26w$ edges; thus, one of the labelled combinatorial maps computed will be equal to that of $\Pi(\Gamma, \alpha)$.

Second case. Let us now assume that ν is an internal node of \mathcal{B} . As above, let α be the arc of \mathcal{B} incident to ν that is the closest to the root ρ . Let β and γ be the arcs different from α incident to ν . Let \mathcal{L}_β and \mathcal{L}_γ be exhaustive lists of bounding graphs for G_β and G_γ , respectively. Intuitively, every pair of bounding graphs in \mathcal{L}_β and \mathcal{L}_γ that are compatible, in the sense that the regions labelled 2 in each of these two graphs are disjoint, will lead to a bounding graph in \mathcal{L}_α . This is the motivating idea to our approach. More precisely, we will enumerate labelled combinatorial maps Π , each of which can be “restricted” to two compatible graphs, which are possible bounding graphs for G_β and G_γ . If these two restrictions lie in \mathcal{L}_α and \mathcal{L}_β , this leads to a graph that is added to \mathcal{L}_α .

We first introduce some terminology. Let Π be a graph properly embedded on \mathcal{C} (possibly non-cellularly), with faces labelled 0, 1, 2, or 3, and with labels on some vertices. Let i, j, k be integers such that $\{i, j, k\} = \{1, 2, 3\}$. We will define a graph embedding $\Pi_{i,j}$ obtained

from Π by somehow “merging” faces i and j . First, for an illustration, refer back to Figures 2 and 3: If Π is the graph embedding depicted on the right of Figure 3, then the configurations shown on the right of Figure 2 correspond, from top to bottom, to $\Pi_{2,3}$, $\Pi_{1,3}$, and $(\Pi_{1,2})^-$ (the latter being the graph $\Pi_{1,2}$ in which each face label 3 is replaced by a 2).

Formally, $\Pi_{i,j}$ is defined as follows. First, let us replace all face labels j by i . Now, for each face f of Π that is homeomorphic to a disk and labelled 0, we do the following. The boundary of f is made of edges of Π ; for the sake of the discussion, let us temporarily label each such edge by the label of the face on the other side of f . If all edges on the boundary of f are all labelled i , then we remove all these edges, and f becomes part of a larger face labelled i . Otherwise, for each maximal subsequence e_1, \dots, e_ℓ of edges along the boundary of f that are all labelled i , we remove each of e_1, \dots, e_ℓ , and replace them with an edge inside f from the source of e_1 to the target of e_ℓ . Finally, we remove all isolated vertices that do not coincide with a singular point of \mathcal{C} , and all vertices in the relative interior of an isolated segment that are incident to two faces with the same label.

For any labelled combinatorial map Π , we denote by Π^- the same map where each label 3 on a face is replaced by a 2. The easy but key properties of this construction are the following:

- (i) Assume that $\Pi_{1,3}$ is a bounding graph for G_β and $(\Pi_{1,2})^-$ is a bounding graph for G_γ . Then $\Pi_{2,3}$ is a bounding graph for G_α .
- (ii) The node ν naturally partitions the edge set of G into three parts, which we denote by E_1 (on the side of α), E_2 (on the side of β), and E_3 (on the side of γ). Assume that G has a sparse proper cellular embedding Γ on \mathcal{C} and that $\Pi = \Pi(\Gamma, E_1, E_2, E_3)$. Then:
 - $\Pi(\Gamma, \alpha) = \Pi(\Gamma, E_1, E_2 \cup E_3) = \Pi_{2,3}$;
 - $\Pi(\Gamma, \beta) = \Pi(\Gamma, E_1 \cup E_3, E_2) = \Pi_{1,3}$;
 - $\Pi(\Gamma, \gamma) = \Pi(\Gamma, E_1 \cup E_2, E_3) = (\Pi_{1,2})^-$.

Property (ii) is, again, illustrated by Figures 2 and 3: If (E_1, E_2, E_3) is the edge partition depicted on Figure 3, then the edge partitions depicted on Figure 2, left, are, respectively, $(E_1, E_2 \cup E_3)$, $(E_1 \cup E_3, E_2)$, and $(E_1 \cup E_2, E_3)$. As shown above, the corresponding partitioning graphs are respectively $\Pi_{2,3}$, $\Pi_{1,3}$, and $\Pi_{1,2}^-$.

If Π is the graph embedding depicted on the right of Figure 3, then the configurations shown on the right of Figure 2 correspond, from top to bottom, to $\Pi_{2,3}$, $\Pi_{1,3}$, and $(\Pi_{1,2})^-$ (the latter being the graph $\Pi_{1,2}$ in which each face label 2 is replaced by a 2).

We compute our exhaustive list \mathcal{L}_α of sparse bounding graphs for G_α as follows. Initially, let this list be empty. Using Lemma 3.3, we enumerate all combinatorial maps Π of graphs with at most $c + 3w$ vertices and $3(74c + 26w)$ edges properly embedded on \mathcal{C} (possibly non-cellularly), with faces labelled 0, 1, 2, or 3, and such that the labels appearing on the vertices are exactly the vertices of the middle set of α , β , or γ (and each label appears exactly once). This takes $(c + w)^{O(c+w)}$ time. Whenever $\Pi_{1,3} \in \mathcal{L}_\beta$ and $(\Pi_{1,2})^- \in \mathcal{L}_\gamma$, we add $\Pi_{2,3}$ to \mathcal{L}_α . Finally, we eliminate duplicates by testing pairwise isomorphism between the labelled combinatorial maps in \mathcal{L}_α , and remove the graphs that are not sparse or contain vertices that bear a label not in the middle set of α .

\mathcal{L}_α contains only sparse bounding graphs for G_α , by (i) above. Moreover, let Γ be a sparse proper cellular graph embedding of G on \mathcal{C} . By sparsity, one of the graphs Π enumerated in the previous paragraph is $\Pi(\Gamma, \nu)$. By definition of \mathcal{L}_β and \mathcal{L}_γ , we have that $\Pi(\Gamma, \beta) \in \mathcal{L}_\beta$ and $\Pi(\Gamma, \gamma) \in \mathcal{L}_\gamma$, so by (ii) above, $\Pi(\Gamma, \alpha) \in \mathcal{L}_\alpha$, which implies that \mathcal{L}_α is exhaustive. ◀

6 Reduction to proper cellular embeddings

► **Proposition 6.1.** *Let \mathcal{C} be a 2-complex with at most c simplices, and G a graph with at most n vertices and edges and branchwidth at most w . Assume that G and \mathcal{C} satisfy the properties of Proposition 3.1. In $c^{O(c)} + O(cn)$ time, one can compute a graph G' , and $c^{O(c)}$ 2-complexes \mathcal{C}_i , such that:*

1. *each \mathcal{C}_i and G' satisfy the properties of Proposition 3.1;*
2. *G' has at most $5cn$ vertices and $5cn$ edges, and branchwidth at most w ;*
3. *each \mathcal{C}_i has size at most c ;*
4. *if, for some i , G' embeds into \mathcal{C}_i , then G embeds into \mathcal{C} ;*
5. *if G embeds into \mathcal{C} , then for some i , G' has a proper cellular embedding into \mathcal{C}_i .*

Sketch of proof. Roughly but not exactly, G' is obtained from G by dissolving every degree-two vertex of G and then subdividing edges $\Theta(c)$ times, and the complexes \mathcal{C}_i are all the 2-complexes “smaller” than \mathcal{C} , obtained by detaching the singular points of \mathcal{C} in all possible ways, removing parts of the isolated segments of \mathcal{C} in all possible ways, and simplifying the topology of the detached surface in all possible ways. It is clear that, if G' embeds into one of these complexes, then G embeds into \mathcal{C} . Conversely, if G embeds into \mathcal{C} , then, by the subdivision process above, G' has a proper embedding into \mathcal{C} , and the only problem is that the faces of G' may fail to be disks. However, by modifying \mathcal{C} using the appropriate choice of operations above, these faces are transformed into disks. ◀

7 Algorithm for bounded branchwidth: Proof of Theorem 1.2

Proof of Theorem 1.2. The proof of this theorem follows directly from the previous propositions. After the preprocessing step (Proposition 3.1), combining Propositions 4.1 and 5.1 give an algorithm that takes as input G and \mathcal{C} and reports one of the following true statements: (i) G has no proper cellular embedding on \mathcal{C} ; (ii) G has an embedding on \mathcal{C} . So, we apply this algorithm to the graph G' and each 2-complex \mathcal{C}_i obtained from Proposition 6.1. If for at least one of these instances, the outcome is (ii), then G has an embedding into \mathcal{C} . Otherwise, G has no embedding into \mathcal{C} . ◀

8 Reduction to bounded branchwidth: Proof of Theorem 1.3

Sketch of proof of Theorem 1.3. This is based on an irrelevant method. We assume that G has large branchwidth. An important fact that we will use is that, if G is embeddable on \mathcal{C} , it has genus $O(c)$.

A **wall** of size $k \times k$ is a subgraph of the $(k \times k)$ -grid obtained by removing alternately the vertical edges of even (resp. odd) x -coordinate in each even (resp. odd) line, and then the degree-one vertices.

We first use any algorithm to approximate the treewidth of G , e.g., Fomin et al. [11, Theorem 1.1]: In polynomial time, we either compute a branch decomposition of small width of G , in which case we are done, or correctly report that the treewidth is large. In the latter case, the result by Chekuri and Chuzhoy [3] implies that there is a large grid minor, which we can compute in randomized polynomial time using the same article, or in deterministic $f(k) \cdot n^2$ time, where f is some computable function and k is the size of the grid [22, Algorithm 4.4]. We have thus computed a subdivision of a large wall.

We then partition this wall into $\Omega(c)$ smaller subwalls W_1, \dots, W_m , where $m = \Omega(c)$, each bounded by a cycle γ_i . We then show that we can assume that there are (intuitively) not too many connections, in G , between two different walls W_i, W_j that avoid the cycles γ_i

and γ_j ; otherwise, by computing these connections, we exhibit a subgraph of G of genus $\Omega(c)$, and deduce that G is not embeddable on \mathcal{C} . Finally, we compute a cycle γ of G , such that one connected component of $G - \gamma$ is planar and contains a subdivision of a smaller, but still large, wall.

We then show, borrowing some ingredients to Kociumaka and Pilipczuk [15, Section 5], that the central vertex of this wall is irrelevant, in the sense that its removal does not affect the embeddability or non-embeddability of the graph into \mathcal{C} . Intuitively, v is surrounded by $\Omega(c)$ concentric cycles in the wall; if $G - v$ is embedded in \mathcal{C} , then two concentric cycles must bound an annulus. The planar part inside the inner cycle can be embedded close to the boundary of the annulus that corresponds to this inner cycle.

Iterating the whole procedure, we obtain a graph of branchwidth polynomial in the size of \mathcal{C} . ◀

References

- 1 Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 641–650, 2008.
- 2 Christoph Buchheim, Markus Chimani, Carsten Gutwenger, Michael Jünger, and Petra Mutzel. Crossings and planarization. In Roberto Tamassia, editor, *Handbook of graph drawing and visualization*. Chapman and Hall, 2006.
- 3 Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM*, 63(5):Article 40, 2016.
- 4 Éric Colin de Verdière. Computational topology of graphs on surfaces. In Jacob E. Goodman, Joseph O’Rourke, and Csaba Toth, editors, *Handbook of Discrete and Computational Geometry*, chapter 23. CRC Press LLC, third edition, 2018.
- 5 Éric Colin de Verdière and Arnaud de Mesmay. Testing graph isotopy on surfaces. *Discrete & Computational Geometry*, 51(1):171–206, 2014.
- 6 Éric Colin de Verdière, Thomas Magnard, and Bojan Mohar. Embedding graphs into two-dimensional simplicial complexes. In *Proceedings of the 34th International Symposium on Computational Geometry (SOCG)*, pages 27:1–27:14, 2018.
- 7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer-Verlag, 2015.
- 8 Tamal K. Dey and Sumanta Guha. Transforming curves on surfaces. *Journal of Computer and System Sciences*, 58:297–325, 1999.
- 9 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608, 2003.
- 10 Jeff Erickson and Kim Whittlesey. Transforming curves on surfaces redux. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1646–1655, 2013.
- 11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):Article 34, 2018.
- 12 John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.
- 13 Ken-ichi Kawarabayashi, Bojan Mohar, and Bruce Reed. A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 771–780, 2008.

- 14 Ken-ichi Kawarabayashi and Bruce Reed. Computing crossing number in linear time. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 382–390, 2007.
- 15 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *Algorithmica*, 81:3655–3691, 2019.
- 16 Francis Lazarus and Julien Rivaud. On the homotopy test on surfaces. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 440–449, 2012.
- 17 Seth M. Malitz. Genus g graphs have pagenumbers $O(\sqrt{g})$. *Journal of Algorithms*, 17:85–109, 1994.
- 18 Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.
- 19 Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.
- 20 Colm Ó Dúnlaing, Colum Watt, and David Wilkins. Homeomorphism of 2-complexes is equivalent to graph isomorphism. *International Journal of Computational Geometry & Applications*, 10:453–476, 2000.
- 21 Maurizio Patrignani. Planarity testing and embedding. In Roberto Tamassia, editor, *Handbook of graph drawing and visualization*. Chapman and Hall, 2006.
- 22 Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.
- 23 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner’s conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004.
- 24 Marcus Schaefer. Toward a theory of planarity: Hanani–Tutte and planarity variants. *Journal of Graph Algorithms and Applications*, 17(4):367–440, 2013.
- 25 John Stillwell. *Classical topology and combinatorial group theory*. Springer-Verlag, New York, second edition, 1993.
- 26 Carsten Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

Efficient Sequential and Parallel Algorithms for Multistage Stochastic Integer Programming Using Proximity

Jana Cslovjceksek ✉

EPFL, Lausanne, Switzerland

Friedrich Eisenbrand ✉

EPFL, Lausanne, Switzerland

Michał Pilipczuk ✉

University of Warsaw, Poland

Moritz Venzin ✉

EPFL, Lausanne, Switzerland

Robert Weismantel ✉

ETH, Zürich, Switzerland

Abstract

We consider the problem of solving integer programs of the form $\min\{c^T x : Ax = b, x \in \mathbb{Z}_{\geq 0}\}$, where A is a multistage stochastic matrix in the following sense: the primal treedepth of A is bounded by a parameter d , which means that the columns of A can be organized into a rooted forest of depth at most d so that columns not bound by the ancestor/descendant relation do not have non-zero entries in the same row. We give an algorithm that solves this problem in fixed-parameter time $f(d, \|A\|_\infty) \cdot n \log^{O(2^d)} n$, where f is a computable function and n is the number of rows of A . The algorithm works in the strong model, where the running time only measures unit arithmetic operations on the input numbers and does not depend on their bitlength. This is the first fpt algorithm for multistage stochastic integer programming to achieve almost linear running time in the strong sense. For two-stage stochastic integer programs, our algorithm works in time $2^{((r+s)\|A\|_\infty)^{O(r(r+s))}} \cdot n \log^{O(rs)} n$, which improves over previous methods both in terms of the polynomial factor and in terms of the dependence on r and s . In fact, for $r = 1$ the dependence on s is asymptotically almost tight assuming the Exponential Time Hypothesis. Our algorithm can be also parallelized: we give an implementation in the PRAM model that achieves running time $f(d, \|A\|_\infty) \cdot \log^{O(2^d)} n$ using n processors.

The main conceptual ingredient in our algorithms is a new proximity result for multistage stochastic integer programs. We prove that if we consider an integer program P , say with a constraint matrix A , then for every optimum solution to the linear relaxation of P there exists an optimum (integral) solution to P that lies, in the ℓ_∞ -norm, within distance bounded by a function of $\|A\|_\infty$ and the primal treedepth of A . On the way to achieve this result, we prove a generalization and considerable improvement of a structural result of Klein for multistage stochastic integer programs. Once the proximity results are established, this allows us to apply a treedepth-based branching strategy guided by an optimum solution to the linear relaxation.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases parameterized algorithm, multistage stochastic programming, proximity

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.33

Related Version Full Version: <https://arxiv.org/abs/2012.11742>

Funding Friedrich Eisenbrand: received funding from the Swiss National Science Foundation within the project Lattice Algorithms and Integer Programming (Nr. 200021-185030).

Michał Pilipczuk: received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 677651).

Robert Weismantel: received support from the Einstein Foundation Berlin.



© Jana Cslovjceksek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 33; pp. 33:1–33:14

Leibniz International Proceedings in Informatics



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



1 Introduction

We consider *integer linear programming* problems

$$\min\{c^\top x : Ax = b, x \in \mathbb{Z}_{\geq 0}\}, \quad (\spadesuit)$$

that are described by the *constraint matrix* $A \in \mathbb{Z}^{n \times m}$, the *linear objective goal* $c \in \mathbb{Z}^m$ and the *right-hand side* vector $b \in \mathbb{Z}^n$. As in this work we only consider linear objective functions, for brevity we use term *integer programming* instead of *integer linear programming*.

While *integer programming* is NP-hard, there are various natural assumptions on the constraint matrix A for which (\spadesuit) is solvable in polynomial time. Famous examples include *totally unimodular* and *bimodular integer programming* [15, 1], integer programs with a *constant number of variables* [20, 17] or *bipartite matching* and *shortest path* problems, see, e.g. [25]. Another example are *block-structured* integer programs in which the constraint matrix exhibits a (recursive) block structure. For instance, in the case of *N-fold* integer programming [7], the removal of a small number of constraints (rows of A) results in decomposing the instance into a large number of small independent subproblems.

We focus on the case where the removal of a few *columns* of A results in a large number of independent subproblems, i.e., on the case where A is *two-stage* or *multistage stochastic*.

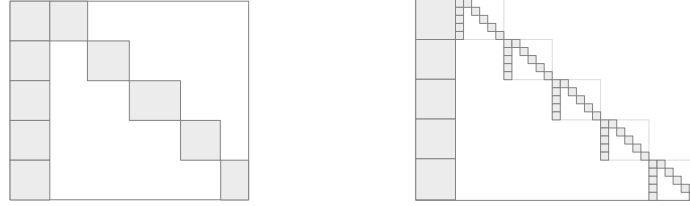


Figure 1 A schematic view of a two-stage stochastic matrix (left panel) and a multistage stochastic matrix (right panel). All non-zero entries are contained in the blocks depicted in grey.

Formally, A is two-stage or (r, s) -*stochastic* (Figure 1, left panel) if after deleting the first r columns the matrix can be decomposed into blocks with at most s columns each. The terminology is borrowed from the field of stochastic integer optimization, a model for discrete optimization under uncertainty. Here, the r “global” variables correspond to a decision made in the first stage, whereas the $\Omega(n)$ blocks involving s variables represent a usually *large* number different *scenarios* that arise in the second stage of stochastic optimization. Two-stage stochastic integer programming has found multiple applications and is a classical topic in optimization, see the survey of Schultz et al. [26] for some examples and algorithms. *Multistage stochastic* integer programming is a generalization of the two-stage variant above obtained by allowing further recursive levels in the block structure (Figure 1, right panel).

The recursive structure in multistage stochastic integer programs can be explained through the notion of the *primal treedepth* of a matrix. The primal treedepth of A , denoted $\text{td}_P(A)$, is the least integer d such that the columns of A can be organized into a rooted forest of depth at most d (called an *elimination forest*) with the following property: for every pair of columns that are not independent – they have non-zero entries in the same row – these columns have to be in the ancestor/descendant relation in the forest. The form presented in Figure 1 can be obtained by ordering the columns as in the top-down depth-first traversal of the elimination forest, and applying a permutation to the rows to form the blocks.

The primal treedepth is a structural parameter that is useful in the design of efficient integer programming solvers. By this, we mean the existence of *fixed-parameter algorithms* for the parameterization by $\text{td}_P(A)$ and $\|A\|_\infty$. For this parameterization, fixed-parameterized tractability can be understood in two ways. *Weak fpt algorithms* have running time of

the form $f(\text{td}_P(A), \|A\|_\infty) \cdot |P|^{\mathcal{O}(1)}$, where f is a computable function and $|P|$ is the total bitlength of the encoding of the input. From *strong fpt algorithms* we require time complexity of the form $f(\text{td}_P(A), \|A\|_\infty) \cdot n^{\mathcal{O}(1)}$, where f is computable and n is the number of rows of the input matrix. Such algorithms work in the model where input numbers occupy single memory cells on which unit-cost arithmetic operations are allowed. Note that thus, the running time is not allowed to depend on the bitlength of the input numbers.

A weak fixed-parameter algorithm for the considered parameterization follows implicitly from the work of Aschenbrenner and Hemmecke [2]. The first to explicitly observe the applicability of primal treedepth to the design of fpt algorithms for integer programming were Ganian and Ordyniak [13], although their algorithm also treats $\|b\|_\infty$ as a parameter besides $\text{td}_P(A)$ and $\|A\|_\infty$. A major development was brought by Koutecký et al. [19], who gave the first strong fpt algorithm, with running time $f(\text{td}_P(A), \|A\|_\infty) \cdot n^3 \log^2 n$. We refer the reader to the joint manuscript of Eisenbrand et al. [10], which comprehensively presents the recent developments in the theory of block-structured integer programming. Corollaries 93 and 96 there discuss the cases of two-stage and multistage stochastic integer programming.

Our contribution. We advance the state-of-the-art of fpt algorithms for two-stage and multistage stochastic integer programming problems by proving the following. Here, n and d respectively denote the number of rows and the primal treedepth of the constraint matrix A .

- A) We give an $f(d, \|A\|_\infty) \cdot n \log^{\mathcal{O}(2^d)} n$ -time algorithm for integer programming (\spadesuit) in the strong sense, where f is a computable function (Theorem 8). This improves upon the currently fastest strong fpt algorithm by Koutecký et al. [19] that is nearly cubic in n .
- B) We provide a $2^{((r+s)\|A\|_\infty)^{\mathcal{O}(r(r+s))}} \cdot n \log^{\mathcal{O}(rs)} n$ -time algorithm for (r, s) -stochastic integer programming, again in the strong sense. This improves upon the currently fastest algorithm that runs in time $2^{(2\|A\|_\infty)^{\mathcal{O}(r^2s+rs^2)}} \cdot n^{\mathcal{O}(1)}$ [10, 18], both in terms of the parametric dependence and in terms of the polynomial factor in the running time.

The algorithmic contributions A and B rely on the following proximity result for integer programs with low primal treedepth. This result can be regarded as the core contribution of this paper, and we believe that it uncovers an important connection between the primal treedepth of A and the solution space of (\spadesuit).

- C) (Proximity) For each optimal solution x^* to the linear relaxation of (\spadesuit) there is an optimal (integral) solution x^\diamond such that $\|x^\diamond - x^*\|_\infty$ is bounded by a computable function of $\text{td}_P(A)$ and $\|A\|_\infty$. (This is proved in Lemma 3.)

This proximity result provides a very simple template for designing fpt algorithms for multistage integer programming. Let us explain it for the case of (r, s) -stochastic IPs. After one has found an optimal fractional solution x^* of the linear relaxation of (\spadesuit), one only has to enumerate the $(2 \cdot f(d, \|A\|_\infty) + 1)^r$ many possible integer assignments for the r stage 1 variables that are within the allowed distance, where $f(d, \|A\|_\infty)$ is the proximity bound provided by Item C. For each of these assignments, the integer program (\spadesuit) decomposes into $\mathcal{O}(n)$ independent sub-problems, each with at most s variables. This results in a $f(r, s, \Delta) \cdot n$ -time algorithm (excluding the time needed for solving the linear relaxation). For multistage-stochastic integer programming, this argument has to be applied recursively.

As for solving the linear relaxation, note that to obtain results A and B we need to be able to solve linear programs with low primal treedepth in near-linear fpt time in the strong sense. This is a non-trivial task. Here we rely on a recent paper, Cslovjeczsek et al. [5] have shown that the dual of the linear programming relaxation of (\spadesuit) can be solved in time $n \log^{\mathcal{O}(2^d)} n$.

By linear programming duality, this provides an algorithm for finding the *optimum value* of the linear relaxation of (\spadesuit) within the required complexity, but for applying the approach presented above, we need to actually compute an optimum fractional solution to (\spadesuit) . While it is likely that the approach of Cslovjcek et al. [5] can be modified so that it outputs such a solution as well, we give a self-contained argument using complementary slackness that applies the results of [5] only as a black-box.

The approach of Cslovjcek et al. [5] is parallelizable, in the sense that the algorithm for solving the linear relaxation of (\spadesuit) can be implemented on a PRAM with n processors so that the running time is $\log^{\mathcal{O}(2^d)} n$, assuming the constraint matrix A is suitably organized on input. As the simple enumeration technique sketched above also can be easily applied in parallel, we obtain the following PRAM counterpart of A and B.

- D) In both cases A and B, we provide algorithms that run in time $f(d, \|A\|_\infty) \cdot \log^{\mathcal{O}(2^d)} n$ and $2^{((r+s)\|A\|_\infty)^{\mathcal{O}(r(r+s))}} \cdot \log^{\mathcal{O}(rs)} n$, respectively, on a PRAM with n processors. For A we assume that the constraint matrix is suitably organized on input.

The proof of Item C relies on a structural lemma of Klein [18], which allows us to bound the ℓ_∞ -norm of the projections of Graver-basis elements to the space of stage 1 variables. In the language of convex geometry, the lemma says the following: if the intersection of integer cones $C_1, \dots, C_m \subseteq \mathbb{Z}^d$ is non-empty, where each generator of each C_i has ℓ_∞ -norm at most Δ , then there is an integer vector $b \in \bigcap_{i=1}^m C_i$ that satisfies $\|b\|_\infty \leq 2^{\mathcal{O}(d\Delta)^d}$. In fact, the original bound of Klein [18] is doubly exponential in d^2 , while we provide a new proof that improves this to a doubly exponential dependence on $d \log d$ only. A direct implication of this is the improvement in the parametric factor reported in B. We also consider some further relaxations of the statement that appear to be important in the proof of Item C.

Related work. The algorithm proposed by Koutecký et al. [19] for multistage stochastic programming relies on *iterative augmentation* using elements of the *Graver basis*, see also [7, 23, 14]. The Graver basis of a matrix A consists of all minimal integer solutions of $Ax = 0$. Here, minimal is w.r.t. the partial order \sqsubseteq of \mathbb{R}^n where $x \sqsubseteq y$ if for each $i \in \{1, \dots, n\}$ we have $|x_i| \leq |y_i|$ and $x_i y_i \geq 0$. Intuitively, a Graver basis element comprises of “single steps” in the lattice of points x satisfying $Ax = 0$. The augmentation framework is to iteratively improve the current solution along directions in the Graver basis. It turns out that in the case of multistage stochastic programs, the ℓ_∞ -norms of the elements of the Graver basis of the constraint matrix A can be bounded by $g(\text{td}_P(A), \|A\|_\infty)$ for some computable function g . This makes the iterative augmentation technique applicable in this setting. It seems that the augmentation framework is however inherently sequential.

Let us note that Koutecký et al. [19] relied on bounds on the function g above due to Aschenbrenner and Hemmecke [2], which only guaranteed computability. Better and explicit bounds on g were later given by Klein [18], see also [10]. Roughly speaking, the proof of Klein [18] shows that $g(d, a)$ is at most d -fold exponential, and it is open whether this bound can be improved to an elementary function.

On a related note, Jansen et al. [16] have very recently given a $2^{2^{\mathcal{O}(s)}} \cdot n^{\mathcal{O}(1)}$ lower bound for $(1, s)$ -stochastic IPs in which all coefficients of the constraint matrix are bounded by a constant in absolute values. This is assuming the Exponential Time Hypothesis. Thus, for $(1, s)$ -stochastic integer programming with bounded coefficients, our result B is almost tight.

While robust and elegant, iterative augmentation requires further arguments to accelerate the convergence to an optimal solution in order to guarantee a good running time. As presented in [10], to overcome this issue one can either involve the bitlength of the input numbers in measuring the complexity, thus resorting to weak fpt algorithms, or reduce this

bitlength using arguments originating in the work of Frank and Tardos [12]. For instance, integer program (\spadesuit) can be solved in time $f(d, \|A\|_\infty) \cdot n^{1+o(1)} \cdot \log^d \|c\|_\infty$, where $d = \text{td}_P(A)$. However, to the best of our knowledge, before this work there was no strong fpt algorithm that would achieve a subquadratic running time dependence on n , even in the setting of two-stage stochastic integer programming.

The setting of N -fold and tree-fold integer programming, which is dual to the setting considered in this work, has received a lot of attention in the literature, see e.g. [3, 5, 7, 10, 14, 24]. Here, we mostly rely on the recent results of Cslovjecsek et al. [5]. They obtained nearly linear-time strong fpt algorithms using an approach quite different from iterative augmentation, which served as a loose inspiration for our work. The key component is a proximity result for integer programs with bounded dual treedepth: they show that if P is an integer program with constraint matrix A , then for every optimal solution x^* to a suitable linear relaxation of P there exists an optimal (integral) solution x^\diamond to P such that $\|x^\diamond - x^*\|_1$ is bounded by a function of $\|A\|_\infty$ and the *dual treedepth* of A (i.e. primal treedepth of A^\top). It follows that if a solution x^* is available, then an optimal integral solution x^\diamond can be found in linear fpt time using dynamic programming, where the bound on $\|x^\diamond - x^*\|$ is used to limit the number of relevant states. This approach requires devising an auxiliary algorithm for solving linear relaxations with bounded dual treedepth in strong fpt time. This is achieved through recursive Laplace dualization using ideas from Norton et al. [22].

Let us stress that our proximity bound provided by Item C requires a different proof using completely different tools than the one obtained for tree-fold integer programs by Cslovjecsek et al. [5]. Note also that our proximity result concerns the standard linear relaxation, whereas the one in [5] holds for the strengthened relaxation, where the blocks are replaced by their integer hulls.

Very recently, Dong et al. [8] proposed a sophisticated interior-point algorithm to approximately solve linear programs whose constraint matrices have primal treewidth t in time $\tilde{O}(nt^2 \cdot \log(1/\varepsilon))$, where ε is an accuracy parameter. Note here that the primal treewidth is bounded by the primal treedepth, so this algorithm in principle could be applied to the linear relaxation of (\spadesuit) . There are two caveats: the algorithm of [8] provides only an approximate solution, and it is unclear whether it can be parallelized. For these reasons we rely on the algorithm of Cslovjecsek et al. [5] through dualization, but exploring the applicability of the work of Dong et al. [8] in our context is an exciting perspective for future work.

Organization. In this paper we focus on presenting the proximity result Item C and deriving algorithmic corollaries. Discussion of solving the linear relaxation as well as full proofs of statements marked with (\heartsuit) , can be found in the full version of this paper, which is available on ArXiv [6].

2 Preliminaries

Model of computation. We assume a real RAM model of computation, where each memory cell stores a real number (of arbitrary bitlength and precision) and arithmetic operations (including rounding) are assumed to be of unit cost. For parallel computation we assume the CRCW PRAM model. As we will be working with sparse matrices, we assume that a matrix is specified on input by a list of its non-zero entries.

(Integer) linear programming. We consider integer programs of the form (\spadesuit) . When replacing the integrality constraint $x \in \mathbb{Z}_{\geq 0}$ by $x \in \mathbb{R}_{\geq 0}$ yields the linear relaxation of (\spadesuit) . We represent a program P as a quadruple $P = (x, A, b, c)$, where x, A, b, c are as in (\spadesuit) . We

denote by $\text{Sol}^{\mathbb{Z}}(P)$ (resp. $\text{Sol}^{\mathbb{R}}(P)$) the set of feasible integral solutions to P (resp. the set of fractional solutions to the linear relaxation to P). Analogously, we denote the set of optimal solutions (with respect to the objective function) by $\text{opt}^{\mathbb{Z}}(P)$ and $\text{opt}^{\mathbb{R}}(P)$ respectively.

Stochastic matrices. We say that a matrix M is *block-decomposable* if there are non-zero block matrices M_1, \dots, M_t such that M can be written as $M = \text{diag}(M_1, \dots, M_t)$. The matrices M, M_1, \dots, M_t need not be square. The *block decomposition* of M is then the unique presentation of M as $M = \text{diag}(M_1, \dots, M_t)$, where the blocks M_1, \dots, M_t are non empty and not block-decomposable.

For nonnegative integers r and s , a matrix A is (r, s) -*stochastic* if the following condition holds: if A' is A with the first r columns removed, then each block in the block decomposition of A' has at most s columns. Equivalently, an (r, s) -stochastic matrix can be written as

$$A = \begin{pmatrix} A_1 & B_1 & & & \\ A_2 & & B_2 & & \\ \vdots & & & \ddots & \\ A_t & & & & B_t \end{pmatrix}, \quad (\diamond)$$

where the blocks A_1, \dots, A_t have r columns and each block B_i has at most s columns. As usual, in (\diamond) and throughout the paper, empty spaces denote blocks filled with zeros. In general, a presentation of matrix A as in (\diamond) is called the *stochastic decomposition* of A . To define the *primal treedepth* $\text{td}_P(A)$ of a matrix A , we first recursively define the *depth* of A :

- if A has no columns, then its depth is 0;
- if A is block-decomposable, then its depth is equal to the maximum among the depths of the blocks in its block decomposition; and
- if A has at least one column and is not block-decomposable, then the depth of A is one larger than the depth of the matrix obtained from A by removing its first column.

The primal treedepth of A is then the smallest integer d , such that the rows and columns of A can be permuted so that the resulting matrix has depth d .

For the remainder of this paper we will assume that matrices of bounded primal treedepth are suitably organized on input. That is, rows and columns are permuted so that the matrix has primal treedepth at most d and is in the block form depicted above. Finding such a permutation can be done in linear fpt time, we discuss this in the full version of the paper.

Graver bases. We collect some basic facts about Graver bases, for a thorough introduction to the theory and its applications we refer to [23, 21]. For an integer matrix A , we write $\ker^{\mathbb{Z}}(A)$ for the set of all integer vectors from $\ker(A)$. The *Graver basis* of A , denoted $\mathcal{G}(A)$, consists of all \sqsubseteq -minimal vectors of $\ker^{\mathbb{Z}}(A)$. We will use the following known bounds on $g_{\infty}(A) := \max_{v \in \mathcal{G}(A)} \|v\|_{\infty}$, the maximum norm of Graver basis elements:

► **Theorem 1** ([9], ∞). *For every integer matrix A with n rows and m columns, we have $g_{\infty}(A) \leq (2n\|A\|_{\infty} + 1)^n$ and $g_{\infty}(A) \leq (2m\|A\|_{\infty} + 1)^m$.*

We will also use the more general bounds for matrices with bounded primal treedepth.

► **Theorem 2** (Lemma 26 of [10]). *There is a computable function $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that for every integer matrix A , $g_{\infty}(A) \leq f(\text{td}_P(A), \|A\|_{\infty})$.*

We note that the proof of Theorem 2 given by Eisenbrand et al. [10] shows that, roughly speaking, $g_{\infty}(A)$ is bounded by a d -fold exponential function of $\|A\|_{\infty}$, where d is the primal treedepth of A .

3 Algorithms

As discussed, our algorithms use two ingredients: proximity results for stochastic integer programs, and algorithms for solving their linear relaxations. In this section we state those ingredients formally and argue how the results claimed in Section 1 follow.

As for proximity, we show that in stochastic integer programs, for every optimal fractional solution there is always an optimal integral solution that is not far, in terms of the ℓ_∞ -norm. Precisely, the following results will be proved in Section 5.

► **Lemma 3.** *There exists a computable function $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ with the following property. Suppose $P = (x, A, b, c)$ is a linear program in the form (\spadesuit) . Then for every optimal fractional solution $x^\star \in \text{Sol}^{\mathbb{R}}(P)$ there exists an optimal integral solution $x^\diamond \in \text{Sol}^{\mathbb{Z}}(P)$ satisfying*

$$\|x^\diamond - x^\star\|_\infty \leq f(\text{depth}(A), \|A\|_\infty).$$

► **Lemma 4.** *Suppose $P = (x, A, b, c)$ is a linear program in the form (\spadesuit) , where A is (r, s) -stochastic for some positive integers r, s . Then for every optimal fractional solution $x^\star \in \text{Sol}^{\mathbb{R}}(P)$ there exists an optimal integral solution $x^\diamond \in \text{Sol}^{\mathbb{Z}}(P)$ satisfying*

$$\|x^\diamond - x^\star\|_\infty \leq 2^{\mathcal{O}(r(r+s)\|A\|_\infty)^{r(r+s)}}.$$

A (r, s) -stochastic matrix has depth at most $r + s$, so Lemma 4 can be seen as a special case of Lemma 3, but provides an explicit bound. For solving the linear relaxation we obtain:

► **Lemma 5** (\spadesuit). *Suppose we are given a linear program $P = (x, A, b, c)$ in the form (\spadesuit) , where A has n rows. Then, in the PRAM model, one can, using n processors and in time $\log^{\mathcal{O}(2^{\text{depth}(A)})} n$, compute an optimal fractional solution to P .*

► **Lemma 6** (\spadesuit). *Suppose we are given an (r, s) -stochastic linear program $P = (x, A, b, c)$ in the form (\spadesuit) , where A has n rows. Then, in the PRAM model, one can, using n processors and in time $2^{\mathcal{O}(r^2+rs^2)} \cdot \log^{\mathcal{O}(rs)} n$, compute an optimal fractional solution to P .*

Again, Lemma 6 differs from Lemma 5 by considering a more restricted class of matrices (i.e., (r, s) -stochastic), but providing better complexity bounds.

We now combine the tools presented above to show the following theorems.

► **Theorem 7** (\spadesuit). *Suppose we are given an (r, s) -stochastic linear program $P = (x, A, b, c)$ in the form (\spadesuit) , where A has n rows. Then, in the PRAM model, one can, using n processors and in time $2^{((r+s)\|A\|_\infty)^{\mathcal{O}(r(r+s))}} \cdot \log^{\mathcal{O}(rs)} n$, compute an optimal integral solution to P .*

Sketch of proof. Apply Lemma 5 to find an optimal fractional solution x^\star . By Lemma 4, there is an optimal integral solution x^\diamond satisfying $\|x^\diamond - x^\star\|_\infty \leq \rho$, where $\rho \in 2^{\mathcal{O}(r(r+s)\|A\|_\infty)^{r(r+s)}}$. In particular, if x_0^\diamond and x_0^\star are the projections of x^\diamond and x^\star onto the first r coordinates, respectively, then $\|x_0^\diamond - x_0^\star\|_\infty \leq \rho$.

Assume the stochastic decomposition (\diamond) of A . For all $\xi \in \mathbb{Z}_{\geq 0}^r$ satisfying $\|\xi - x_1^\star\|_\infty \leq \rho$ and all $i \in \{1, \dots, t\}$, let us consider the integer program $P_i(\xi)$ defined as:

$$\min\{c_i^\top x_i : B_i x_i = b_i - A_i \xi, x_i \geq 0\},$$

where b_i, c_i, x_i are suitable restrictions of b, c, x to entries corresponding to rows or columns of B_i . It follows that

$$\text{opt}^{\mathbb{Z}}(P) = \min \left\{ c_0^\top \xi + \sum_{i=1}^t \text{opt}^{\mathbb{Z}}(P_i(\xi)) : \xi \geq 0 \text{ is integral and } \|\xi - x_1^\star\|_\infty \leq \rho \right\},$$

where c_0 is the projection of c onto the first r coordinates. Therefore, it suffices to iterate through all integral vectors $\xi \geq 0$ satisfying $\|\xi - x_1^*\| \leq \rho$ one by one – of which there are at most $(2\rho + 1)^r$ many – and for each of them solve all the programs $P_i(\xi)$ in parallel, by assigning to $P_i(\xi)$ as many processors as the number of rows of B_i .

It remains to argue how each of the programs $P_i(\xi)$ is going to be solved efficiently. For this, we may apply the same approach. Namely, we use Lemma 5 to find an optimal fractional solution x_i^* of $P_i(\xi)$, and using Lemma 4 again we can argue that there exists an optimal integral solution x_i° of $P_i(\xi)$ that satisfies $\|x_i^\circ - x_i^*\|_\infty \leq \rho$. Now B_i has at most s columns, so there are only at most $(2\rho + 1)^s$ candidates for an integral vector x_i° satisfying the above, and they can be checked one by one. The overall running time analysis follows easily from the bounds provided by Lemma 4 and Lemma 5; we leave the details to the reader. ◀

The same basic idea, but applied recursively, yields the following.

► **Theorem 8** (\asymp). *There is a computable function $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that the following holds. Suppose we are given a linear program $P = (x, A, b, c)$ in the form (\spadesuit), where A has n rows. Then, in the PRAM model, one can using n processors and in time $f(\text{depth}(A), \|A\|_\infty) \cdot \log^{\mathcal{O}(2^{\text{depth}(A)})} n$ compute an optimal integral solution to P .*

4 A stronger Klein bound

In this section we discuss a stronger variant of a structural result of Klein [18] which we will need for our proximity bounds in the next section.

► **Theorem 9** (Stronger Klein bound, \asymp). *Let $T_1, \dots, T_n \subseteq \mathbb{Z}^d$ be multisets of integer vectors of ℓ_∞ -norm at most Δ such that their respective sums are almost the same in the following sense: there is some $b \in \mathbb{Z}^d$ and a positive integer ϵ such that*

$$\left\| \sum_{v \in T_i} v - b \right\|_\infty < \epsilon \quad \text{for all } i \in \{1, \dots, n\}.$$

There exists a function $f(d, \Delta) \in 2^{\mathcal{O}(d\Delta)^d}$ such that the following holds. Assuming $\|b\|_\infty > \epsilon \cdot f(d, \Delta)$, one can find nonempty submultisets $S_i \subseteq T_i$ for all $i \in \{1, \dots, n\}$, and a vector $b' \in \mathbb{Z}^d$ satisfying $\|b'\|_\infty \leq f(d, \Delta)$, such that

$$\sum_{v \in S_i} v = b' \quad \text{for all } i \in \{1, \dots, n\}.$$

Theorem 9 strengthens the original formulation of Klein [18, Lemma 2] in various aspects. First, the formulation of Klein required all the vectors to be nonnegative. Second, the argument of Klein yields a bound on $f(d, \Delta)$ that is doubly exponential in d^2 , our proof improves this dependence to doubly exponential in $d \log d$. Finally, we allow the sums of the respective multisets to differ by some slack parameter ϵ , while in the original setting of Klein all sums need to be exactly equal. This last aspect will prove essential in the proof of our proximity bound, while the second is primarily used for improving the parametric factor in the running time.

The full proof of Theorem 9 relies on polyhedral techniques and is rather lengthy. However, using only the original formulation due to Klein and the pigeonhole principle, there is a short proof that achieves the third aspect of our improvement, i.e. that we may allow the sums of respective multisets to differ by some slack parameter. As this is central for the next section, we prove only this part and defer the full proof of Theorem 9 to the full version of the paper.

Proof of a simpler variant of Theorem 9. Specifically, we show the following statement:

Let T_1, \dots, T_n be multisets of vectors in $\mathbb{Z}_{\geq 0}^d$ of ℓ_∞ norm at most Δ such that there is some $b \in \mathbb{Z}^d$ and $\epsilon \in \mathbb{N}$ such that

$$\left\| \sum_{v \in T_i} v - b \right\|_\infty < \epsilon \quad \text{for all } i \in \{1, \dots, n\}.$$

Then there is a function $g(d, \Delta)$ such that provided $\|b\|_\infty > \epsilon \cdot g(d, \Delta)$, there exist nonempty submultisets $S_i \subseteq T_i$ for $i \in \{1, \dots, n\}$ and a vector $b' \in \mathbb{Z}_{\geq 0}^d$ satisfying $\|b'\|_\infty \leq g(d, \Delta)$ such that

$$\sum_{v \in S_i} v = b' \quad \text{for all } i \in \{1, \dots, n\}.$$

In fact, one can take $g(d, \Delta) = 2(f(d, \Delta) + 1)^{d+1}$ where $f(d, \Delta)$ is the bound from [18].

To prove this, we first add vectors belonging to $\{0, 1\}^d$ to each multiset T_i so that

$$\sum_{v \in \tilde{T}_i} v = b + \epsilon \cdot \mathbf{1} \quad \text{for all } i \in \{1, \dots, n\},$$

where the \tilde{T}_i are the resulting multisets and $\mathbf{1}$ is the all-ones vector. Clearly, this can be achieved by adding to each multiset T_i at most 2ϵ vectors from $\{0, 1\}^d$.

Assume that $\|b\|_\infty > 2\epsilon \cdot (f(d, \Delta) + 1)^{d+1}$ where $f(d, \Delta)$ is the original bound from [18]. Since the multisets \tilde{T}_i all sum up to $b + \epsilon \cdot \mathbf{1}$ exactly, we can use the original formulation of [18] to infer that there are submultisets $S_i^1 \subseteq \tilde{T}_i$, for all $\{1, \dots, n\}$, and a vector $b_1 \in \mathbb{Z}_{\geq 0}^d$ with $\|b_1\|_\infty \leq f(d, \Delta)$ such that

$$\sum_{v \in S_i^1} v = b_1 \quad \text{for all } i \in \{1, \dots, n\}.$$

Since the multisets $\tilde{T}_i - S_i^1$ sum up to $b + \epsilon \cdot \mathbf{1} - b_1$, we can iteratively find nonempty submultisets $S_i^2 \subseteq \tilde{T}_i - S_i^1, \dots, S_i^k \subseteq \tilde{T}_i - (S_i^1 \cup \dots \cup S_i^k)$ and vectors b_1, \dots, b_k of ℓ_∞ -norm bounded by $f(d, \Delta)$ such that

$$\sum_{v \in S_i^j} v = b_j \quad \text{for all } j \in \{2, \dots, k\} \text{ and } i \in \{1, \dots, n\}.$$

Since we assumed that $\|b\|_\infty > 2\epsilon \cdot (f(d, \Delta) + 1)^{d+1}$, we can continue the above procedure until $k > 2\epsilon \cdot (f(d, \Delta) + 1)^d$. Note that there are at most $(f(d, \Delta) + 1)^d$ integral and nonnegative vectors of ℓ_∞ norm at most $f(d, \Delta)$. Therefore, by pigeonhole principle there exists $b' \in \mathbb{Z}_{\geq 0}^d$ with $\|b'\|_\infty \leq f(d, \Delta)$ and a set of indices J of size $2\epsilon + 1$ such that $\sum_{v \in S_i^j} v = b'$ for all $j \in J$ and $i \in \{1, \dots, n\}$. For each $i \in \{1, \dots, n\}$, one of these multisets $S_i^j \subseteq \tilde{T}_i$ for $j \in J$ does not contain any of the (at most) 2ϵ vectors we have added to T_i to obtain \tilde{T}_i . Thus, for each $i \in \{1, \dots, n\}$ we can find a nonempty submultisets $S_i \subseteq T_i$ satisfying

$$\sum_{v \in S_i} v = b'.$$

Since $\|b'\|_\infty \leq f(d, \Delta) \leq g(d, \Delta)$, this concludes the proof. \blacktriangleleft

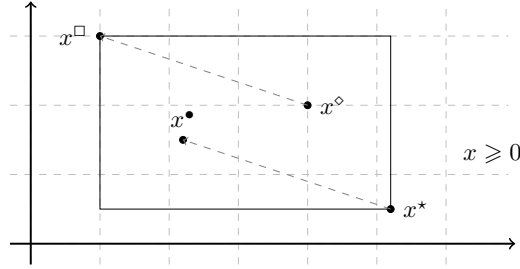
5 Proximity

The goal of this section to prove a very general theorem, Theorem 12, that will imply Lemma 3 and Lemma 4, see Corollaries 14 and 15. To facilitate the discussion of proximity, let us introduce the following definition.

► **Definition 10.** Let $P = (x, A, b, c)$ be a linear program in the form (\spadesuit). The proximity of P , denoted $\text{proximity}_\infty(P)$, is the infimum of reals $\rho \geq 0$ satisfying the following: for every fractional solution $x^\star \in \text{Sol}^\mathbb{R}(P)$ and integral solution $x^\square \in \text{Sol}^\mathbb{Z}(P)$, there is an integral solution $x^\diamond \in \text{Sol}^\mathbb{Z}(P)$ such that

$$\|x^\diamond - x^\star\|_\infty \leq \rho \quad \text{and} \quad x^\diamond - x^\star \subseteq x^\square - x^\star.$$

The condition $x^\diamond - x^\star \subseteq x^\square - x^\star$ is equivalent to saying that x^\diamond is contained in the axis parallel box spanned by x^\star and x^\square , see Figure 2.



■ **Figure 2** $x^\diamond - x^\star \subseteq x^\square - x^\star$, x^\diamond is in the rectangle spanned by x^\square and x^\star .

Comparing to earlier work, for instance [4, 11], this notion of proximity is independent of the optimization goal. However, it can also be used to bound the distance of optimal fractional solutions to optimal integral solutions.

► **Lemma 11** (\times). Suppose $P = (x, A, b, c)$ is a linear program in the form (\spadesuit). Then for every optimal fractional solution x^\star to P there is an optimal integral solution x^\diamond to P satisfying

$$\|x^\diamond - x^\star\|_\infty \leq \text{proximity}_\infty(P).$$

Sketch of proof, see Figure 2. Let x^\square any optimal integral solution. By our definition of proximity there is x^\diamond with $\|x^\diamond - x^\star\|_\infty \leq \text{proximity}_\infty(P)$ and $x^\diamond - x^\star \subseteq x^\square - x^\star$. It can be easily checked that if $c^\top x^\square < c^\top x^\diamond$, then $x^\bullet := x^\star + x^\square - x^\diamond$ is feasible and $c^\top x^\bullet < c^\top x^\star$, contradicting the optimality of x^\star . Thus, x^\diamond is an optimal integral solution. ◀

For the remainder of this section we adopt the following notation. Suppose that A has a stochastic decomposition (\diamond). Let x_0, x_1, \dots, x_t be the partition of the vector of variables x so that $x_0 \in \mathbb{R}^r$ corresponds to the columns of matrices A_1, \dots, A_t , while $x_i \in \mathbb{R}^s$ corresponds to the columns of B_i , for each $i \in \{1, \dots, t\}$. Finally, partition b into b_1, \dots, b_t so that b_i corresponds to the rows of A_i and B_i respectively. Thus, $\text{Sol}^\mathbb{R}(P)$ takes the form:

$$\begin{aligned} A_i x_0 + B_i x_i &= b_i & \text{for all } i \in \{1, \dots, t\}, \\ x_i &\geq 0 & \text{for all } i \in \{0, 1, \dots, t\}. \end{aligned}$$

For each $i \in \{1, \dots, t\}$, we define

$$P_i = \left(\begin{pmatrix} x_0 \\ x_i \end{pmatrix}, D_i, b_i, 0 \right) \quad \text{with} \quad D_i := (A_i \ B_i). \quad (1)$$

We observe that

$$\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \end{pmatrix} \in \text{Sol}^{\mathbb{R}}(P) \quad \text{if and only if} \quad \begin{pmatrix} x_0 \\ x_i \end{pmatrix} \in \text{Sol}^{\mathbb{R}}(P_i) \quad \text{for all } i \in \{1, \dots, t\}.$$

This decomposition of the constraint matrix is the key to our main technical result.

► **Theorem 12** (Composition Theorem). *Suppose $P = (x, A, b, c)$ is a linear program in the form (\spadesuit) , where A admits a stochastic decomposition (\diamond) . Adopt the notation presented above and let k be the number of columns of each of the matrices A_1, \dots, A_t . Further, let*

$$\gamma := \max_{1 \leq i \leq t} g_{\infty}(D_i) \quad \text{and} \quad \rho := \max_{1 \leq i \leq t} \text{proximity}_{\infty}(P_i).$$

Then

$$\text{proximity}_{\infty}(P) \leq 3k\gamma\rho \cdot f(k, \gamma)$$

where $f(k, \gamma)$ is the bound provided by Theorem 9.

By substituting $f(k, \gamma)$ with the bound of Theorem 9, we get $\text{proximity}_{\infty}(P) \leq \rho \cdot 2^{\mathcal{O}(k\gamma)^k}$. To derive the promised bounds on the proximity we use the following simple lemma.

► **Lemma 13** (\times). *Let $P = (x, A, b, c)$ be a linear program in the form (\spadesuit) where A has m columns. Then*

$$\text{proximity}_{\infty}(P) \leq (m\|A\|_{\infty})^{m+1}.$$

Sketch of proof. Given a feasible fractional solution x^* and an integral vector x^{\square} , we can consider the ILP on m variables defined by constraints $Ax = b$ and $x - x^* \sqsubseteq x^{\square} - x^*$. By the classic theorem of Cook et al. [4], there is an integral solution whose ℓ_{∞} distance from x^* is at most m times the largest sub-determinant. It remains to apply the Hadamard bound. ◀

► **Corollary 14.** *Let $P = (x, A, b, c)$ be a linear program in the form (\spadesuit) , where A is (r, s) -stochastic. Then*

$$\text{proximity}_{\infty}(P) \leq 2^{\mathcal{O}(r(r+s)\|A\|_{\infty})^{r(r+s)}}.$$

Proof. Recalling the definition of P_i in (1) and using that the matrix A is (r, s) -stochastic, we see that the constraint matrix D_i of P_i has at most $r + s$ columns with entries bounded by $\|A\|_{\infty}$. Using Lemma 13 and Theorem 1 respectively, we get

$$g_{\infty}(D_i) \leq (2(r + s)\|A\|_{\infty} + 1)^{r+s} \quad \text{and} \quad \text{proximity}_{\infty}(P_i) \leq ((r + s)\|A\|_{\infty})^{r+s+1}.$$

By Theorem 12 we obtain the claimed bound on $\text{proximity}_{\infty}(A)$. ◀

Applying the same idea recursively yields the following.

► **Corollary 15** (\times). *There is a computable function $h: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that for every linear program $P = (x, A, b, c)$ in the form (\spadesuit) , we have*

$$\text{proximity}_{\infty}(P) \leq h(\text{td}_P(A), \|A\|_{\infty}).$$

5.1 Proof of Theorem 12

We now use our strengthening of the lemma of Klein [18], Theorem 9, to prove Theorem 12.

Proof of Theorem 12. Consider any $x^* \in \text{Sol}^{\mathbb{R}}(P)$ and $x^\square \in \text{Sol}^{\mathbb{Z}}(P)$. Let $x^\diamond \in \text{Sol}^{\mathbb{Z}}(P)$ be an integral solution such that $x^\diamond - x^* \sqsubseteq x^\square - x^*$ and subject to the condition that $\|x^\diamond - x^*\|_1$ is minimized. Our goal is to show that then $\|x^\diamond - x^*\|_\infty \leq 3k\gamma\rho \cdot f(k, \gamma)$, where $f(\cdot, \cdot)$ is the function given by Theorem 9.

Observe that if there existed a non-zero vector $u \in \ker^{\mathbb{Z}}(A)$ such that $u \sqsubseteq x^* - x^\diamond$, then we would have that $x^\diamond + u \in \text{Sol}^{\mathbb{Z}}(P)$, $(x^\diamond + u) - x^* \sqsubseteq x^\diamond - x^* \sqsubseteq x^\square - x^*$, and the ℓ_1 distance from x^* to $x^\diamond + u$ would be strictly smaller than to x^\diamond . This would contradict the choice of x^\diamond . Therefore, it is sufficient to show the following: if $\|x^\diamond - x^*\|_\infty$ is larger than $3k\gamma\rho \cdot f(k, \gamma)$, then there exists a non-zero vector $u \in \ker^{\mathbb{Z}}(A)$ such that $u \sqsubseteq x^* - x^\diamond$.

To this end, for all $i \in \{1, \dots, t\}$, restrict x^* and x^\diamond to the variables of P_i as follows:

$$\tilde{x}_i^* := \begin{pmatrix} x_0^* \\ x_i^* \end{pmatrix} \in \text{Sol}^{\mathbb{R}}(P_i) \quad \text{and} \quad \tilde{x}_i^\diamond := \begin{pmatrix} x_0^\diamond \\ x_i^\diamond \end{pmatrix} \in \text{Sol}^{\mathbb{Z}}(P_i).$$

By the definition of proximity, for all $i \in \{1, \dots, t\}$ there are integral solutions

$$\tilde{x}_i \in \text{Sol}^{\mathbb{Z}}(P_i) \quad \text{with} \quad \|\tilde{x}_i - \tilde{x}_i^*\|_\infty \leq \text{proximity}_\infty(P_i) \leq \rho \quad \text{and} \quad \tilde{x}_i - \tilde{x}_i^* \sqsubseteq \tilde{x}_i^\diamond - \tilde{x}_i^*.$$

Since \tilde{x}_i and \tilde{x}_i^\diamond are both integral solutions to P_i , we have $\tilde{x}_i - \tilde{x}_i^\diamond \in \ker^{\mathbb{Z}}(A_i \ B_i)$ and we can decompose this vector into a multiset G_i of Graver elements. That is, G_i is a multiset consisting of sign compatible (i.e., belonging to the same orthant) elements of $\mathcal{G}(D_i)$ with $\tilde{x}_i - \tilde{x}_i^\diamond = \sum_{g \in G_i} g$. Note that the first k entries of vectors $\tilde{x}_1, \dots, \tilde{x}_t$ correspond to the same k variables of P , but they may differ for different $i \in \{1, \dots, t\}$. For a vector w , let $\pi(w)$ be the projection onto the first k entries of w .

Let $\pi(G_i)$ be the multiset that includes a copy of $\pi(g)$ for each $g \in G_i$. By the definition of \tilde{x}_i^* and \tilde{x}_i^\diamond , we have $\pi(\tilde{x}_i^*) = \pi(\tilde{x}_j^*)$ and $\pi(\tilde{x}_i^\diamond) = \pi(\tilde{x}_j^\diamond)$ for all $i, j \in \{1, \dots, t\}$. From this, for all $i \in \{1, \dots, t\}$,

$$\left\| \sum_{x \in \pi(G_i)} x - \pi(\tilde{x}_1^* - \tilde{x}_1^\diamond) \right\|_\infty = \left\| \pi(\tilde{x}_i) - \underbrace{\pi(\tilde{x}_1^*)}_{=\pi(\tilde{x}_i^*)} + \underbrace{\pi(\tilde{x}_1^\diamond) - \pi(\tilde{x}_i^\diamond)}_{=0} \right\|_\infty \leq \|\tilde{x}_i - \tilde{x}_i^*\|_\infty \leq \rho.$$

Thus, Theorem 9 is applicable for $d = k$, $\Delta = \gamma$, $\epsilon = \rho$ and $b = \pi(\tilde{x}_1^* - \tilde{x}_1^\diamond)$. Note that for each $i \in \{1, \dots, t\}$ and $g \in G_i$, we have $\|g\|_\infty \leq \gamma$. We now distinguish between two cases:
Case 1: $\|\pi(\tilde{x}_1^* - \tilde{x}_1^\diamond)\|_\infty > \rho \cdot f(k, \gamma)$.

By Theorem 9, there exist nonempty submultisets $S_1 \subseteq \pi(G_1), \dots, S_t \subseteq \pi(G_t)$ such that

$$\sum_{x \in S_i} x = \sum_{x \in S_j} x \quad \text{for all } i, j \in \{1, \dots, t\}.$$

Define a vector u in the following way. For all $i \in \{1, \dots, t\}$, let $\hat{G}_i \subseteq G_i$ be submultisets with $\pi(\hat{G}_i) = S_i$ and set $\tilde{u}_i := \sum_{g \in \hat{G}_i} g \in \ker^{\mathbb{Z}}(D_i)$. Observe that vectors $\pi(\tilde{u}_i)$ are equal for all $i \in \{1, \dots, t\}$. This allows us to define u as the vector obtained by combining all the \tilde{u}_i , so that projecting u to the variables of P_i yields \tilde{u}_i , for each $i \in \{1, \dots, t\}$. Note that since multisets \hat{G}_i are nonempty, u is a non-zero vector. Also $u \in \ker^{\mathbb{Z}}(A)$, since $\tilde{u}_i \in \ker^{\mathbb{Z}}(D_i)$ for all $i \in \{1, \dots, t\}$. Further, we have $u \sqsubseteq x^* - x^\diamond$, because for all $i \in \{1, \dots, t\}$,

$$\tilde{u}_i = \sum_{g \in \hat{G}_i} g \sqsubseteq \tilde{x}_i - \tilde{x}_i^\diamond \sqsubseteq \tilde{x}_i^* - \tilde{x}_i^\diamond.$$

Thus, u contradicts the minimality of $\|x^\diamond - x^\star\|_1$. We move to the second case.

Case 2: $\|\pi(\tilde{x}_1^\star - \tilde{x}_1^\diamond)\|_\infty \leq \rho \cdot f(k, \gamma)$.

Since we have $\|\pi(\tilde{x}_i - \tilde{x}_i^\diamond) - \pi(\tilde{x}_1^\star - \tilde{x}_1^\diamond)\|_\infty \leq \rho$ for all $i \in \{1, \dots, t\}$, we have

$$\|\pi(\tilde{x}_i - \tilde{x}_i^\diamond)\|_\infty \leq \rho \cdot f(k, \gamma) + \rho \leq 2\rho \cdot f(k, \gamma) \quad \text{for all } i \in \{1, \dots, t\}.$$

Suppose for a moment that for some $i \in \{1, \dots, t\}$, there exists an element $g \in G_i$ with $\pi(g) = 0$. Then by putting zeros on all the other coordinates, we can extend g to a vector $u \in \ker^{\mathbb{Z}}(A)$ which satisfies $u \sqsubseteq x^\star - x^\diamond$. As g is non-zero, so is u , hence u satisfies all the requested properties. Hence, from now on we may assume that no multiset G_i contains an element g with $\pi(g) = 0$. It follows that

$$|G_i| = |\pi(G_i)| \leq \left\| \sum_{x \in \pi(G_i)} x \right\|_1 \leq k \left\| \sum_{x \in \pi(G_i)} x \right\|_\infty = k \|\pi(\tilde{x}_i - \tilde{x}_i^\diamond)\|_\infty \leq 2k\rho \cdot f(k, \gamma).$$

Since $\|g\|_\infty \leq \gamma$ for every element $g \in G_i$, we infer that

$$\|\tilde{x}_i - \tilde{x}_i^\diamond\|_\infty \leq \left\| \sum_{g \in G_i} g \right\|_\infty \leq \gamma |G_i| \leq 2k\gamma\rho \cdot f(k, \gamma).$$

By combining this with $\|\tilde{x}_i - \tilde{x}_i^\star\|_\infty \leq \rho$, we get

$$\|\tilde{x}_i^\diamond - \tilde{x}_i^\star\|_\infty \leq \|\tilde{x}_i^\diamond - \tilde{x}_i\|_\infty + \|\tilde{x}_i - \tilde{x}_i^\star\|_\infty \leq 2k\gamma\rho \cdot f(k, \gamma) + \rho \leq 3k\gamma\rho \cdot f(k, \gamma).$$

This implies that $\|x^\diamond - x^\star\| \leq 3k\gamma\rho \cdot f(k, \gamma)$ and concludes the proof. \blacktriangleleft

References

- 1 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 1206â€“1219, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055473.
- 2 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Found. Comput. Math.*, 7(2):183–227, 2007. doi:10.1007/s10208-005-0174-1.
- 3 Lin Chen and Daniel Marx. Covering a tree with rooted subtrees: Parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 2801â€“2820, USA, 2018. Society for Industrial and Applied Mathematics.
- 4 W. Cook, A. M. H. Gerards, A. Schrijver, and É. Tardos. Sensitivity theorems in integer linear programming. *Mathematical Programming*, 34(3):251–264, 1986. doi:10.1007/BF01582230.
- 5 Jana Cslovjecssek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1666–1681. SIAM, 2021. doi:10.1137/1.9781611976465.101.
- 6 Jana Cslovjecssek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity, 2020. arXiv:2012.11742.
- 7 Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008. In Memory of George B. Dantzig. doi:10.1016/j.disopt.2006.06.006.
- 8 Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. *CoRR*, abs/2011.05365, 2020. arXiv:2011.05365.

- 9 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.49.
- 10 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019. arXiv:1904.01361.
- 11 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1), 2019. doi:10.1145/3340322.
- 12 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 13 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. In *30th Conference on Artificial Intelligence, AAAI 2016*, pages 710–716. AAAI Press, 2016. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12432>.
- 14 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N -fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013. doi:10.1007/s10107-011-0490-y.
- 15 Alan J Hoffman and Joseph B Kruskal. Integral boundary points of convex polyhedra. In *Linear Inequalities and Related Systems.(AM-38), Volume 38*, pages 223–246. Princeton University Press, 2016.
- 16 Klaus Jansen, Kim-Manuel Klein, and Alexandra Lassota. The double exponential runtime is tight for 2-stage stochastic ILPs. *CoRR*, abs/2008.12928, 2020. arXiv:2008.12928.
- 17 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.
- 18 Kim-Manuel Klein. About the complexity of two-stage stochastic IPs. In *Proceedings of the 21st International Conference, IPCO 2020*, volume 12125 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2020. doi:10.1007/978-3-030-45771-6_20.
- 19 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.85.
- 20 H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. URL: <http://www.jstor.org/stable/3689168>.
- 21 Jesús De Loera, Raymond Hemmecke, and Matthias Kppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*. Society for Industrial and Applied Mathematics, USA, 2012.
- 22 Carolyn Haiht Norton, Serge A. Plotkin, and Éva Tardos. Using separation algorithms in fixed dimension. *J. Algorithms*, 13(1):79–98, 1992. doi:10.1016/0196-6774(92)90006-X.
- 23 S. Onn. *Nonlinear Discrete Optimization: An Algorithmic Theory*. European Mathematical Society Publishing House, 2010. URL: <https://books.google.ch/books?id=wzAGMQAACAAJ>.
- 24 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics*, European Mathematical Society, 2010.
- 25 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science, 2003.
- 26 R. Schultz, L. Stougie, and M. H. van der Vlerk. Two-stage stochastic integer programming: a survey. *Statistica Neerlandica*, 50(3):404–416, 1996. doi:10.1111/j.1467-9574.1996.tb01506.x.

Modular Counting of Subgraphs: Matchings, Matching-Splittable Graphs, and Paths




Radu Curticapean   

Basic Algorithm Research Copenhagen (BARC), IT University of Copenhagen, Denmark

Holger Dell   

Goethe Universität Frankfurt, Germany

Basic Algorithm Research Copenhagen (BARC), IT University of Copenhagen, Denmark

Thore Husfeldt   

Basic Algorithm Research Copenhagen (BARC), IT University of Copenhagen, Denmark
Lund University, Sweden

Abstract

We systematically investigate the complexity of counting subgraph patterns *modulo fixed integers*. For example, it is known that the *parity* of the number of k -matchings can be determined in polynomial time by a simple reduction to the determinant. We generalize this to an $n^{f(t,s)}$ -time algorithm to compute modulo 2^t the number of subgraph occurrences of patterns that are s vertices away from being matchings. This shows that the known polynomial-time cases of subgraph *detection* (Jansen and Marx, SODA 2015) carry over into the setting of *counting modulo 2^t* . Complementing our algorithm, we also give a simple and self-contained proof that counting k -matchings modulo odd integers q is $\text{Mod}_q\text{W}[1]$ -complete and prove that counting k -paths modulo 2 is $\oplus\text{W}[1]$ -complete, answering an open question by Björklund, Dell, and Husfeldt (ICALP 2015).

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Counting complexity, matchings, paths, subgraphs, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.34

Related Version *Full Version*: <https://arxiv.org/abs/2107.00629>

Funding Supported by VILLUM Foundation grant 16582.

1 Introduction

The last two decades have seen the development of several complexity dichotomies for pattern counting problems in graphs, including full classifications for counting *subgraphs*, *induced subgraphs*, and *homomorphisms* from fixed computable pattern classes \mathcal{H} . The input to such problems is a *pattern* graph $H \in \mathcal{H}$ and an unrestricted *host* graph G ; the task is to count the relevant occurrences of H in G . Depending on \mathcal{H} , these problems are known to be either polynomial-time solvable or $\#\text{W}[1]$ -hard when parameterized by $|V(H)|$. The latter rules out polynomial-time algorithms under the complexity assumption $\text{FPT} \neq \#\text{W}[1]$.

In this paper, we focus on counting *subgraphs* from any fixed graph class \mathcal{H} . On the positive side, given a pattern graph $H \in \mathcal{H}$ whose smallest vertex-cover has size $\text{vc}(H)$ and an n -vertex host graph G , there are known $O(n^{\text{vc}(H)+1})$ time algorithms [40, 29, 9] to count subgraphs of G that are isomorphic to H : First, find a minimum vertex-cover C of H using exhaustive search. Then, iterate over all possible embeddings f of $H[C]$ into G and count the possible extensions of $G[f(C)]$ to a full copy of H . Complementing this algorithm, an almost matching running time lower bound of $n^{\Omega(\text{vc}(H)/\log \text{vc}(H))}$ under the exponential-time hypothesis (ETH) is also known [8]. Thus, assuming ETH or $\text{FPT} \neq \#\text{W}[1]$, the problem



© Radu Curticapean, Holger Dell, and Thore Husfeldt;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 34; pp. 34:1–34:17



Leibniz International Proceedings in Informatics

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

$\#\text{Sub}(\mathcal{H})$ of counting subgraphs from a fixed class \mathcal{H} is polynomial-time solvable if and only if the vertex-cover numbers (or equivalently, the maximum matching sizes) of the graphs in \mathcal{H} are bounded by a constant. The rightmost column of Figure 1 visualizes this situation.

Turning from counting to the problem $\text{Sub}(\mathcal{H})$ of *detecting* subgraphs from fixed classes \mathcal{H} , the picture is less clear. Evidence points at three strata of complexity: Define the *matching-split number* of H to be the minimum number of vertices whose deletion turns H into a matching, that is, a graph of maximum degree 1. Jansen and Marx [25] show that, if this number is bounded in a graph class \mathcal{H} , then $\text{Sub}(\mathcal{H})$ is polynomial-time solvable. For classes \mathcal{H} of bounded tree-width, it is known [34, 1, 19] that the problem $\text{Sub}(\mathcal{H})$ is fixed-parameter tractable when parameterized by $|V(H)|$. For pattern classes \mathcal{H} of unbounded tree-width, it is conjectured that $\text{Sub}(\mathcal{H})$ is $\text{W}[1]$ -hard – so far, this hardness has only been established for cliques, bicliques [30], grids [6], and less natural graph classes. The leftmost column of Figure 1 visualizes the situation.





We propose to study an intermediate setting between decision and counting, namely, counting subgraph patterns *modulo fixed integers* $q \in \mathbf{N}$. Modular counting has a tradition in classical complexity theory, where the complexity classes Mod_qP for $q \in \mathbf{N}$ capture problems that ask to count accepting paths of polynomially time-bounded non-deterministic Turing machines modulo q . In particular, the class Mod_2P (better known as $\oplus\text{P}$) plays a central role in the proof of Toda’s theorem [38]. Several (partial) classification results for frameworks of modular counting problems are known; this includes homomorphisms to fixed graphs [16, 20, 21, 26, 18], constraint satisfaction problems [15, 22], and Holant problems [11].

Figure 1 summarizes our understanding. If the vertex-cover number is bounded, the polynomial-time algorithms (regions 7 and 8) follow from the algorithm for $\#\text{Sub}(\mathcal{H})$ described above and require no further attention. Our paper is concerned with the remaining regions 1–6.

As argued above, matchings play a central role in decision and counting, so it is natural that they reprise their role in modular subgraph counting: On the positive side, there are known polynomial-time algorithms for counting matchings of a given size modulo fixed powers of two. (For bipartite graphs and counting modulo 2, this essentially follows from the fact that determinant and permanent coincide modulo 2.) On the negative side, if q is not a power of two, counting matchings modulo q is known to be Mod_pP -complete for any odd prime p dividing q . We establish a parameterized analogue of this fact: Let $\text{Mod}_q\text{W}[1]$ be the class of parameterized problems that are fpt-reducible to counting k -cliques modulo q . We show that counting k -matchings (that is, sets of k pairwise disjoint edges) in graphs modulo fixed odd primes $q \in \mathbf{N}$ is $\text{Mod}_q\text{W}[1]$ -hard. In our proof, modular counting allows us to sidestep the algebraic machinery from previous works [9, 7, 8], resulting in a surprisingly simple and self-contained argument.

► **Theorem 1.** *For any integer $q \in \mathbf{N}$ containing an odd prime factor p , counting k -matchings modulo q is $\text{Mod}_p\text{W}[1]$ -hard under Turing fpt-reductions and admits no $n^{o(k/\log k)}$ time algorithm under ETH.*

Known arguments from Ramsey theory (see [10, Section 5]) extend Theorem 1 from matchings to $\#\text{Sub}(\mathcal{H}) \bmod q$ for any hereditary class \mathcal{H} of unbounded vertex-cover number. This suggests that modular subgraph counting may only become tractable when the modulus is a power of two. Indeed, we show that patterns of matching-split number s can be counted modulo $q = 2^t$ in time $n^{O(t4^s)}$. To prove this, we follow the general idea of the bounded vertex-cover number algorithm for $\text{Sub}(\mathcal{H})$ outlined before, and we reduce to counting matchings modulo powers of two. This however requires us to overcome technical complications to avoid unwanted cancellations. Overall, we obtain:

				This paper		
		<i>grids</i> 		1	2	<i>W[1]-hard</i>
		<i>bicliques</i> 				
tree-width	FPT		<i>paths</i>	3	4	
matching-split number	Polynomial time			5	<i>matchings</i> 	
vertex-cover number				7	8	
	decision	counting mod 2^t		counting mod odd q		counting

■ **Figure 1** An overview over known results and our new results. The columns correspond, from left to right, to the problem types $\text{Sub}(\mathcal{H})$, $\#\text{Sub}(\mathcal{H}) \bmod 2^t$, $\#\text{Sub}(\mathcal{H}) \bmod q$ for $q \neq 2^t$, or $\#\text{Sub}(\mathcal{H})$; our results are depicted in the two middle columns. The rows correspond, from bottom to top, to requiring \mathcal{H} to have bounded vertex-cover number, matching-split number, tree-width, or no requirement at all. The complexity along each row is monotone: By Lemma 4, decision is no harder than modular counting, and modular counting trivially is no harder than counting. Our results are depicted in the middle two columns: Regions 1 and 2 are Lemma 5. Region 5 is Theorem 2. Regions 7 and 8 already follow from [40]. The point in region 6 is Theorem 1, and the point in region 3 is Theorem 3. We view the hardness of these points as evidence to conjecture their enclosing regions to be hard, see Conjecture 14.

► **Theorem 2.** *There is an algorithm that, given a graph H of matching-split number $s \in \mathbb{N}$ and an n -vertex graph G , computes the number of H -isomorphic subgraphs of G modulo 2^t in time $n^{O(t4^s)}$.*

We complement this result in two ways: First, we observe that $\oplus\text{Sub}(\mathcal{H})$ is $\oplus\text{W}[1]$ -complete for pattern classes \mathcal{H} of unbounded tree-width; this follows directly from previous hardness proofs for $\#\text{Sub}(\mathcal{H})$. More interestingly, we establish the $\oplus\text{W}[1]$ -completeness of counting k -paths modulo 2 in undirected graphs, thus solving an open problem from [3], where this problem was considered in the context of Hamiltonian cycle detection, following [4].

► **Theorem 3.** *Counting k -paths modulo 2 is $\oplus\text{W}[1]$ -complete.*

This result adds to a rich range of previous work on the k -path problem, and is of interest outside our framework. Bodlaender [5] and Monien [32] showed that *finding* a k -path is fixed-parameter tractable. In contrast, Flum and Grohe [17] showed that *exactly counting* k -paths is $\#\text{W}[1]$ -hard. Nevertheless, Arvind and Raman [2] showed that *approximately counting* k -paths, which corresponds to computing the most significant bit(s) of the number of k -paths, is fixed-parameter tractable. Our Theorem 3 suggests that the *least* significant bit of the number of k -paths is hard to compute. This is surprising, because some of the most influential fpt-algorithms for finding a k -path work over characteristic 2, based on the group algebra framework introduced by Koutis [28].

Let us conclude with a general remark on the techniques used in this paper: Recent works successfully exploited a connection between subgraph counts and (linear combinations of) homomorphism counts to obtain algorithms and hardness results [8, 35, 14, 36, 37]. For

example, the number of k -matchings in a graph G is a linear combination of homomorphism counts from $f(k)$ fixed graphs. Insights on the complexity of counting the homomorphisms occurring in this linear combination then lead to complexity results for counting k -matchings. This connection however does not readily transfer to modular counting, as the relevant linear combinations (which involve rational coefficients) may be *undefined* modulo p . We therefore prove Theorems 1–3 using more combinatorial approaches.

2 Preliminaries

Unless otherwise stated, we consider finite, undirected, simple graphs without self-loops.

Subgraph problems

A *homomorphism* from graph H to graph G is a mapping $\varphi: V(H) \rightarrow V(G)$ such that $\{\varphi(u), \varphi(v)\} \in E(G)$ for each $\{u, v\} \in E(H)$. An *embedding* is an injective homomorphism, and we let $\text{Emb}(H, G)$ denote the set of embeddings from H to G . An *isomorphism* is a bijective homomorphism, and an *automorphism* is an isomorphism from H to itself. The set of all automorphisms of H is called $\text{Aut}(H)$, and forms a group when endowed with function composition \circ .

We let $\text{Sub}(H, G)$ be the set of all H -subgraphs of G , that is, the set of all H' with $V(H') \subseteq V(G)$ and $E(H') \subseteq E(G)$ such that H' is isomorphic to H . This terminology fixes the possible confusion about isomorphic copies of subgraphs: For example, there is exactly one K_k -subgraph in K_k , but there are $k!$ embeddings. The *subgraph problem* Sub is given a pair (H, G) to decide whether G has at least one H -subgraph. The *subgraph counting problem* $\#\text{Sub}$ is given a pair (H, G) to determine the number of H -subgraphs in G .

For a graph class \mathcal{H} , we write $\#\text{Sub}(\mathcal{H})$ for the restricted problem where the input (H, G) is promised to satisfy $H \in \mathcal{H}$. For $q \in \mathbf{Z}_{\geq 2}$, the *modular subgraph counting problem* $\#\text{Sub}(\mathcal{H}) \bmod q$ is the problem to compute the number of H -subgraphs modulo q . In the special case with $q = 2$, we write $\oplus\text{Sub}$.

It will be useful to consider *colorful* subgraph problems, where G is H -colored, that is, there is a given homomorphism $c: V(G) \rightarrow V(H)$. Due to the homomorphism property, we allow edges $\{u, v\} \in E(G)$ only if the corresponding colors satisfy $\{c(u), c(v)\} \in E(H)$. A subgraph H' of an H -colored graph G is *vertex-colorful* if c is bijective on $V(H')$. Let $\text{VertexColorfulSub}(H, G)$ be the set of vertex-colorful subgraphs H' for which c is an isomorphism from H' to H . The corresponding computational problems are defined analogously to the uncolored case; the input consists of a graph G together with an H -coloring c .

Background from complexity theory

A *parameterized counting problem* is a pair (f, κ) of functions $f, \kappa: \{0, 1\}^* \rightarrow \mathbf{N}$ where κ is computable. A *parsimonious fpt-reduction* from a parameterized counting problem (f, κ) to a parameterized counting problem (g, ι) is a function R with the following properties: (i) $f(x) = g(R(x))$ for all $x \in \{0, 1\}^*$, (ii) $\iota(R(x))$ is bounded by a computable function in $\kappa(x)$, and (iii) the reduction is computable in time $h(\kappa(x)) \text{poly}(|x|)$ for some computable function h . A *Turing fpt-reduction* may query the oracle multiple times for instances whose parameter is bounded by a function of the input parameter, and combine the query answers in fpt-time to produce the correct output. Moreover, reductions can also be *randomized*, in which case we require that their error probability is bounded by a small constant.

The *exponential-time hypothesis* (ETH) postulates the existence of some $\varepsilon > 0$ such that no algorithm solves n -variable 3-CNF formulas in time $O(2^{\varepsilon n})$. We write for short that 3-CNF-SAT does not have $2^{o(n)}$ -time algorithms, and we also disallow bounded-error randomized algorithms.

Modular counting

For our purposes, we define the class $\text{Mod}_q\text{W}[1]$ as the class of all parameterized problems (f, κ) with $f: \Sigma^* \rightarrow \{0, \dots, q-1\}$ such that (f, κ) has a parsimonious fpt-reduction to the problem of counting k -cliques modulo q . For $q = 2$, it was shown in [3] that all problems in $\text{W}[1]$ admit randomized fpt-reductions to problems in $\oplus\text{W}[1]$. Another result [39, Lemma 2.1] yields the corresponding generalization for all $q > 2$. We use the following analogous proposition for the vertex-colorful subgraph problem, proven in the full version.

► **Lemma 4.** *For any integer $q \geq 2$, there is a randomized Turing fpt-reduction from the problem VertexColorfulSub to the problem $\#\text{VertexColorfulSub} \bmod q$. On input (H, G) , the reduction only queries instances with the same pattern H .*

Our work relies on the following hardness result for parameterized modular subgraph counting, which follows easily from known results on the colorful subgraph decision problem [13, 31]. See the full version for a proof.

► **Lemma 5.** *Let \mathcal{H} be a graph family of unbounded tree-width and let q be an integer with $q \geq 2$. Then $\#\text{VertexColorfulSub}(\mathcal{H}) \bmod q$ parameterized by $k = |E(H)|$ is $\text{Mod}_q\text{W}[1]$ -hard under parsimonious fpt-reductions. Moreover, if ETH is true, then the problem does not have an algorithm running in time $n^{o(k/\log k)}$, where $n = |V(G)|$.*

3 Hardness of counting k -matchings

In this section, we prove Theorem 1. We first establish $\text{Mod}_q\text{W}[1]$ -hardness of the problem $\#\text{ColMatch} \bmod q$ for odd $q \geq 3$: Given a graph G with an edge-coloring $c: E(G) \rightarrow \mathcal{C}$ for some set of colors \mathcal{C} with $|\mathcal{C}| = k$, this problem asks to count modulo q the edge-colorful matchings in G . These are the matchings that use each color in \mathcal{C} exactly once.

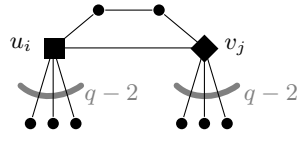
► **Lemma 6.** *For any fixed integer p with odd prime factor q , the problem $\#\text{ColMatch} \bmod p$ is $\text{Mod}_q\text{W}[1]$ -hard under parsimonious fpt-reductions and has no $n^{o(k/\log k)}$ time algorithm under ETH.*

Proof. The class \mathcal{H} of all 3-regular graphs has unbounded tree-width, and hence by Lemma 5, the problem $\#\text{VertexColorfulSub}(\mathcal{H}) \bmod q$ is $\text{Mod}_q\text{W}[1]$ -hard and hard under ETH. We reduce it to $\#\text{ColMatch} \bmod q$, implying the hardness of $\#\text{ColMatch} \bmod p$. Let $H \in \mathcal{H}$ and G be the input for the reduction with $k = |V(H)|$ and H -colored G , and let $\{V_a : a \in V(H)\}$ be the color classes of G , with edge-sets $E_{a,b}(G)$ for $ab \in E(H)$. Using the gadgets from Figure 2, we construct a graph G' :

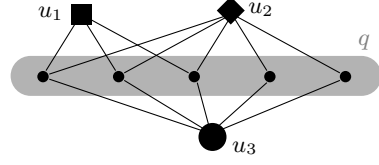
1. Each vertex $u \in V(G)$ is replaced by three vertices u_1, u_2 , and u_3 . We insert a *consistency gadget* Q_u at these vertices by adding q gadget vertices, connecting u_2 and u_3 to all gadget vertices, and u_1 to the first $(q+1)/2$ gadget vertices. For $S \subseteq \{u_1, u_2, u_3\}$, let m_S count the matchings in Q_u that match precisely S ; it can be checked that

$$m_S \equiv_q \begin{cases} 1 & \text{if } S \text{ is } \emptyset \text{ or } \{u_1, u_2, u_3\}, \\ 0 & \text{if } S \text{ is } \{u_2\}, \{u_3\}, \text{ or } \{u_2, u_3\}. \end{cases} \quad (1)$$

We explicitly ignore the other three cases for S , as they will not be relevant.



(a) The AND-gadget, shown here for $q = 5$. In general, the upper u_i, v_j -path always has 3 edges; both *external vertices* u_i and v_j have $q - 2$ neighboring leaves. If exactly 0 or 1 external vertices are removed, this graph has 0 edges modulo q ; if both vertices are removed, the graph has 1 edge.



(b) The consistency gadget contains q gadget vertices, shown here for $q = 5$. The number of matchings of size 0 and 3 equals 1 modulo q , and if u_1 is deleted, the number of non-empty edge-colorful matchings equals 0 modulo q .

■ **Figure 2** The two gadgets used in the proof of Lemma 6.

2. For $\{a, b\} \in E(H)$, suppose that a is the j th vertex incident to b , and that b is the i th vertex incident to a . For each edge $\{u, v\} \in E_{a,b}(G)$ with $u \in V_a$ and $v \in V_b$, we insert an AND-gadget A_{uv} at $\{u_i, v_j\}$. Then for any set $S \subsetneq \{u_i, v_j\}$, the number of edges in $A_{uv} - S$ is divisible by q , whereas $A_{uv} - \{u_i, v_j\}$ has exactly one edge.
3. The edge-colors of G' are defined as follows: For $u \in V_a$ with $a \in V(H)$, we assign color (CONS, a, i) to all edges of Q_u incident to vertex u_i , for $i \in \{1, 2, 3\}$. For each $ab \in E(H)$, we assign color (AND, ab) to all edges in AND-gadgets between edges in $E_{a,b}(G)$. Overall, we have $k' = 3k + 3k/2$ colors.

Every H -copy F in G induces a set \mathcal{M}_F of colorful matchings in G' . We describe this set in the following, show that $|\mathcal{M}_F| \equiv_q 1$, and that \mathcal{M}_F and $\mathcal{M}_{F'}$ are disjoint for $F \neq F'$.

- For each $v \in V(F)$, match all of $\{v_1, v_2, v_3\}$ within Q_v . For fixed v , the number of possible matchings in Q_v is $m_{\{v_1, v_2, v_3\}} \equiv_q 1$ by (1). Let \mathcal{Q}_F denote the set of all matchings that can be obtained by the previous step. Since matchings can be chosen independently for distinct Q_v , we obtain $|\mathcal{Q}_F| \equiv_q 1^{|V(F)|} \equiv_q 1$.
- Any $M \in \mathcal{Q}_F$ can be extended to several colorful matchings by choosing one edge from each color (AND, ab) for $\{a, b\} \in E(H)$. For each $\{u, v\} \in E(F)$, the AND-gadget A_{uv} has exactly one such edge, while the other AND-gadgets of color (AND, ab) have 0 such edges modulo q . Hence, the number edges of color (AND, ab) that can extend M is 1 modulo q . This implies that the overall number r_M of matchings extending M into a colorful matching is also $r_M \equiv_q 1^{|E(F)|} \equiv_q 1$.

Overall, every H -copy F induces $\sum_{M \in \mathcal{Q}_F} r_M \equiv_q \sum_{M \in \mathcal{M}_F} 1 \equiv_q 1$ colorful matchings, so we indeed have $|\mathcal{M}_F| \equiv_q 1$. We also observe from the construction that $\mathcal{M}_F \cap \mathcal{M}_{F'} = \emptyset$ for distinct H -copies F and F' . In the full version, we use properties of the gadgets to prove that colorful matchings $M \notin \bigcup_F \mathcal{M}_F$ cancel modulo q .

▷ **Claim 7.** The number of colorful matchings M that are not contained in \mathcal{M}_F for any H -copy F is divisible by q .

Overall, we have shown that the number of H -copies in G and the number of colorful matchings in G' agree modulo q . As G' can be computed in polynomial time and the parameter is increased only by a constant factor, the claimed hardness results follow. ◀

To prove Theorem 1, it suffices to give an fpt-reduction from $\#\text{ColMatch mod } q$ to counting k -matchings modulo q . This is achieved by a standard inclusion-exclusion argument that can be found in the full version.

4 Counting matching-splittable subgraphs modulo 2^t

In this section, we prove Theorem 2 by describing an $n^{O(t^4)}$ -time algorithm for counting modulo 2^t the subgraphs of matching-split number s . Our algorithm builds upon known algorithms for the decision and counting versions of subgraph problems; we first review their underlying ideas and sketch our algorithm for Theorem 2.

Counting subgraphs of bounded vertex-cover number

The basic structure of our algorithm is similar to a known $O(n^{s+1})$ time algorithm [40, 29, 9] for counting embeddings from H to G if H has a vertex-cover $S \subseteq V(H)$ of size $s \in \mathbb{N}$. Counting embeddings is sufficient for counting subgraph copies, as we can first compute the number $\#Aut(H)$ of automorphisms on H as $\#Emb(H, H)$, and then use

$$\#Sub(H, G) = \frac{\#Emb(H, G)}{\#Aut(H)}. \quad (2)$$

Given (H, G) with $h = |V(H)|$ and $n = |V(G)|$, the algorithm for computing $\#Emb(H, G)$ first finds a minimum vertex-cover S of H in time $h^{O(s)}$; then $I := V(H) \setminus S$ is an independent set. Then the algorithm enumerates all partial embeddings f from $H[S]$ to G , which takes time at most $n^{O(s)}$. Finally, for each f , it remains to count all functions $g: I \rightarrow V(G)$ that extend f to a full embedding from H to G . We observe that g extends f to a full embedding if and only if every vertex $u \in I$ maps via g to a vertex $v = g(u) \in V(G) \setminus f(S)$ that satisfies the *neighborhood constraint* $N_G(v) \cap f(S) \supseteq f(N_H(u))$. Counting functions g with this property can be achieved (in a not completely obvious way) with dynamic programming; we only need to know the number of vertices $v \in V(G) \setminus f(S)$ that have a specific neighborhood $N_G(v) \cap f(S)$, and for each f , there are at most 2^s different possible such neighborhoods. Overall, in $n^{O(s)}$ time, we can compute the number $\#Emb(H, G)$.

Detecting subgraphs of bounded matching-split number

Jansen and Marx [25] extend the above approach and obtain an $n^{O(s)}$ time algorithm for the *decision* problem $Sub(H, G)$ when H has matching-split number s . In this case, we consider a *splitting set* S of size s instead of a vertex-cover, that is, the graph $M = H - S$ may have isolated edges besides isolated vertices. Now the idea is to not only classify the vertices $v \in V(G) \setminus f(S)$ by their neighborhoods $N_v = N_G(v) \cap f(S)$, but to also classify the edges $\{u, v\} \in E(G - f(S))$ by their neighborhoods $\{N_u, N_v\}$. It then remains to find a matching in $G - f(S)$ that has as many isolated vertices and isolated edges as $H - S$, such that these vertices and edges satisfy the neighborhood constraints in $f(S)$. Jansen and Marx achieve this by reduction to a colored matching problem.

Our algorithm

In our algorithm for Theorem 2, we need to overcome two challenges:

- (a) Since counting embeddings is algorithmically more straight-forward than counting subgraphs, we would like to count embeddings and divide by the number of automorphisms $\#Aut(H)$ as in Equation (2). However, since we are counting modulo 2^t , the number $\#Aut(H) \bmod 2^t$ may be 0, and so the division in Equation (2) is impossible. (In fact, even even numbers $\#Aut(H)$ have no inverse modulo 2^t .)
- (b) When mimicking Jansen and Marx's detection algorithm, we cannot just *count* the relevant matchings in $G - f(S)$, since counting perfect matchings is $\#P$ -hard.

Most of our effort focuses on overcoming (a): In Section 4.1, we show that every graph H of matching-split number s has a splitting set R of size $O(s^2)$ that remains rigid under automorphisms, i.e., any automorphism f of H must satisfy $f(R) = R$. In Section 4.2, we show how to compute $\# \text{Sub}(H, G)$ if such a rigid splitting set R for H is given. Rather than counting H -embeddings and attempting a division by $\# \text{Aut}(H)$, we use the rigidity of R to keep track of the automorphisms of H in a more explicit way.

To overcome (b), we use a determinant-based algorithm [23] to compute the Hafnian over a polynomial ring modulo 2^t . We then reduce our constrained matching counting problem to computing such Hafnians. This part of the algorithm can be found in the full version.

4.1 Rigidizing the splitting set

Let H be a graph with a splitting set S of size s , and let $M = H - S$ be the remaining graph of maximum degree 1; we speak of M as a matching, even though it may contain isolated vertices. An automorphism f of H may map a vertex $v \in S$ in the splitting set to $f(v) \notin S$. We show that if a splitting set of size s exists then there is also a *rigid* splitting set R of size $O(s^2)$, i.e., such that every $f \in \text{Aut}(H)$ satisfies $f(R) = R$. In fact, the following algorithm can find such a set R .

Algorithm Rigidize(H) *Given a graph H of matching-split number s , this algorithm computes a rigid splitting set $R \subseteq V(H)$ of size $O(s^2)$.*

- R1** (Find small splitting set.) Using brute-force, compute a set $S \subseteq V(H)$ of size s such that $H - S$ is a matching.
- R2** (Extend it to neighbors of low-degree vertices.) Let $D \subseteq V(H)$ be the set of all vertices whose degree in H is at most $s + 1$. Set $T := S$. While there is an edge $\{u, v\}$ with $u \in T \cap D$ and $v \in \bar{T}$, add v to T .
- R3** (Refine it.) Set $R := T$. For each component C of $H[T \cap D]$ with at most two vertices, remove $V(C)$ from R .

The following lemma captures useful properties of Rigidize. See the full version for a proof.

► **Lemma 8.** *The algorithm Rigidize runs in time $h^{O(s)}$ where $h = |V(H)|$, and the output set $R \subseteq V(H)$ has the properties that $|R| \leq O(s^2)$, that $H - R$ is a matching, and that every $f \in \text{Aut}(H)$ satisfies $f(R) = R$.*

4.2 Counting subgraphs with rigid splitting sets

We use the rigid splitting set R from Lemma 8 to compute the number of times H occurs as a subgraph modulo a power of two. As a subroutine, we use an algorithm for counting colored matchings modulo a power of two in a setting involving particular “color demands”.

► **Definition 9.** *Let G be a graph, let C be a finite set of colors, and let $c: V(G) \cup E(G) \rightarrow 2^C$ be a function that labels each vertex and edge with a subset of C . For any matching M , let $I(M)$ be the set of its isolated vertices. For a coloring $c_M: I(M) \cup E(M) \rightarrow C$, the colored matching (M, c_M) is permissible if $c_M(t) \in c(t)$ holds for all $t \in I(M) \cup E(M)$.*

Color demands are functions $D_I, D_E: C \rightarrow \mathbb{N}$. The pair (M, c_M) satisfies the demands D_I, D_E if, for each $i \in C$, the graph M contains exactly $D_I(i)$ isolated vertices v with $c_M(v) = i$ and exactly $D_E(i)$ edges with $c_M(e) = i$. Let $\mathcal{M}(G, c, D_I, D_E)$ be the set of all permissible matchings (M, c_M) that satisfy the demand D .

As shown in the full version, we obtain the following algorithm as a corollary to Hirai and Namba’s algorithm [23] for computing the Hafnian over polynomial rings modulo 2^t .

► **Lemma 10.** *Given a graph G , permissible colors $c: V(G) \cup E(G) \rightarrow 2^C$, color demands $D_I, D_E: C \rightarrow \mathbf{N}$, and $t \in \mathbf{N}_{\geq 1}$, there is an algorithm that computes the number $|\mathcal{M}(G, c, D_I, D_E)| \bmod 2^t$ in time $n^{O(t|C|)}$.*

Before we state the main algorithm, we introduce some basic group-theoretic notation. Let R be a splitting set of H that satisfies $f(R) = R$ for all $f \in \text{Aut}(H)$. Let G be a graph and let $S \subseteq V(G)$ be a set with $|S| = |R|$. For an embedding $\sigma \in \text{Emb}(H[R], G[S])$ and an automorphism $\varphi \in \text{Aut}(H)$, we note that the function $\sigma \circ (\varphi|_R)$ is again an embedding in $\text{Emb}(H[R], G[S])$. Indeed, we view this operation as a right-action of the group $\text{Aut}(H)$ on the set $\text{Emb}(H[R], G[S])$. We call two embeddings $\sigma, \sigma' \in \text{Emb}(H[R], G[S])$ *equivalent* if there exists $\varphi \in \text{Aut}(H)$ such that $\sigma' = \sigma \circ (\varphi|_R)$; this clearly defines an equivalence relation. The equivalence class $\sigma \text{Aut}(H)$ is called the *orbit* of σ under $\text{Aut}(H)$. All orbits have the same size. Let E_S be a set of representatives for each orbit, that is, a maximal set of mutually non-equivalent embeddings in $\text{Emb}(H[R], G[S])$.

We are ready to state the modular counting algorithm for s -matching-splittable subgraphs.

Algorithm ModCount(H, G, t) *Given an s -matching-splittable graph H , a host graph G , and an integer $t \geq 2$, this algorithm computes the number $\#\text{Sub}(H, G) \bmod 2^t$.*

- C1** (Compute rigid splitting set.) Call $\text{Rigidize}(H)$ to compute the set R .
- C2** (Reduce to counting colored matchings.) For each $S \subseteq V(G)$ with $|S| = |R|$ (that is, a possible image of R) and each representative embedding $\sigma \in E_S$ from $H[R]$ to $G[S]$, we construct an instance $(G - S, c_\sigma, D_I, D_E)$ of colored matching with demands and use Lemma 10 to obtain the number $|\mathcal{M}(G - S, c_\sigma, D_I, D_E)| \bmod 2^t$:
 - a.** (Set permitted colors.) Let $C = 2^R \cup \binom{2^R}{1} \cup \binom{2^R}{2}$. For each vertex $v \in V(G) \setminus S$, let $N_v \subseteq R$ be the vertices of R that hit $N_G(v) \cap S$ under σ , that is, $N_v = \sigma^{-1}(N_G(v) \cap S)$. Define $c_\sigma(v) = \{N : N \subseteq N_v\}$. Moreover, for each $\{u, v\} \in E(G - S)$, define $c_\sigma(\{u, v\}) = \{\{N, N'\} : N \subseteq N_u, N' \subseteq N_v\}$.
 - b.** (Make demands.) The demands $D_I, D_E: C \rightarrow \mathbf{N}$ depend only on H and R . For each $N \subseteq R$, we let $D_I(N)$ be the number of isolated vertices v in $H - R$ whose neighborhood in H satisfies $N_H(v) \cap R = N$. Moreover, for all $N, N' \subseteq R$, we let $D_E(\{N, N'\})$ be the number of edges $\{u, v\} \in E(H - R)$ with $\{N_H(u) \cap R, N_H(v) \cap R\} = \{N, N'\}$.
- C3** (Sum up.) Output the sum modulo 2^t of all integers returned by the queries in C2.

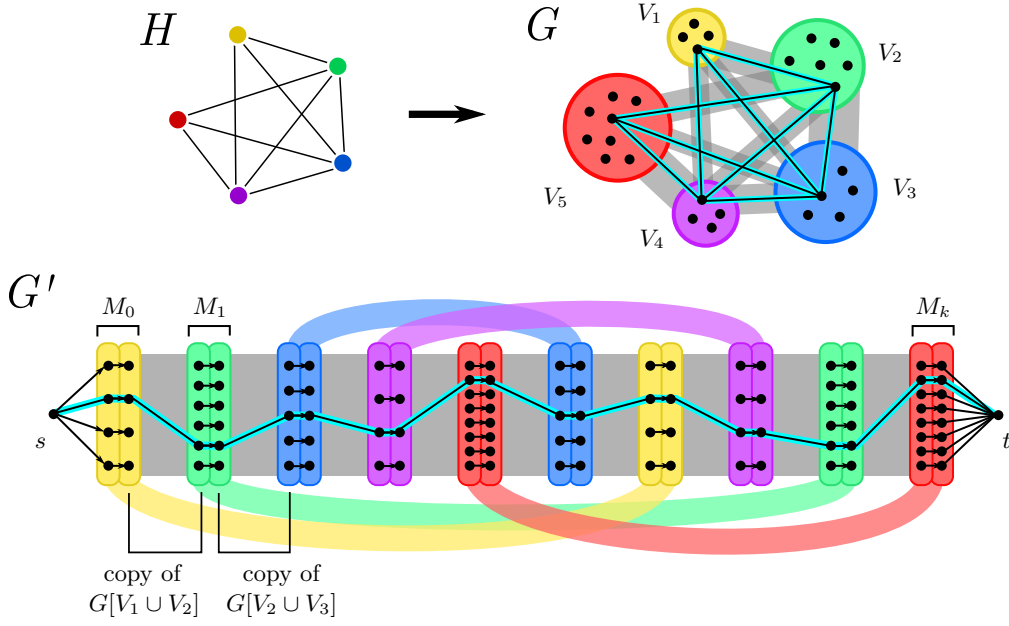
In the full version, we prove that ModCount satisfies the properties stated in Theorem 2.

5 Hardness of counting paths modulo two

In this section, we prove Theorem 3, that counting k -paths modulo 2 is $\oplus W[1]$ -hard. We first formally introduce this and some intermediate problems.

The *length* of a path is the number of its edges. For a graph G and vertices $s, t \in V(G)$, an s, t -*path* is a simple path from s to t . For a computable, strictly increasing function $f: \mathbf{N} \rightarrow \mathbf{N}$, we define f -Flexible Path to be the problem that is given (G, s, t, k) to decide whether there exists any s, t -path in G whose length ℓ satisfies $k \leq \ell \leq f(k)$. When id denotes the identity function, then Path (also known as k -Path or LONGEST PATH) is defined as id -Flexible Path. We similarly define Directed f -Flexible Path and Directed Path for directed graphs, and we define the counting and parity versions of these problems in the canonical manner.

We start our reduction at the vertex-colorful subgraph problem $\oplus \text{VertexColorfulSub}(\mathcal{H})$ for a class \mathcal{H} of unbounded tree-width, which is $\oplus W[1]$ -hard by Lemma 5. The class we choose consists of connected, almost 4-regular graphs without non-trivial automorphisms;



■ **Figure 3** The construction of the graph G' from the graphs H and G . The homomorphism from G to H is indicated by colors. A colorful H -subgraph H' in G and the canonical path corresponding to H' in G' are highlighted in turquoise. Gadget edges are hinted at as semi-transparent curves. Except for (half of the) gadget edges, all edges are oriented from left to right.

here, we say that a graph is *almost 4-regular* if it can be obtained from a 4-regular graph by deleting one edge. The core part of the reduction is in Lemma 11, where we reduce to counting paths of somewhat flexible length in a directed graph (modulo 2). From there, we reduce to the familiar k -path problem in undirected graphs using standard tricks.

► **Lemma 11.** *For any class \mathcal{H} of connected, almost 4-regular graphs without non-trivial automorphisms, there is a computable, strictly increasing function f such that there is parsimonious polynomial-time fpt-reduction from $\oplus\text{VertexColorfulSub}(\mathcal{H})$ to $\oplus\text{Directed } f\text{-Flexible Path}$.*

Proof. Let $H \in \mathcal{H}$ and G be the undirected graphs that are given as input, where G is given with disjoint color classes V_u for $u \in V(H)$. Let $k = |E(H)|$. Since H is connected and almost 4-regular, it has an Eulerian path $u_0, u_1, \dots, u_{k-1}, u_k$ such that $E(H) = \{\{u_i, u_{i+1}\} : i \in \{0, \dots, k-1\}\}$. Every vertex of H appears exactly twice on the Eulerian path, and u_0 and u_k are the two different degree-3 vertices of H . Our goal is to construct a directed graph G' , such that $\oplus\text{VertexColorfulSub}(H, G) = \oplus\text{Directed } f\text{-Flexible Path}(G')$ holds for a suitable f .

Intuitively, the graph G' “visits” every color class V_{u_i} of G two times according to the Eulerian path in H . Before we give a formal construction, we give an overview; see also Figure 3. Essentially, the graph G' is a sequence of directed bipartite graphs B_0, \dots, B_ℓ whose edges are all directed from left to right. For a bipartite graph B , we write $L(B)$ and $R(B)$ for its left and right part, respectively. We have $R(B_j) = L(B_{j+1})$ for all $j \in \{0, \dots, \ell-1\}$. Each B_j is either a perfect matching M_i or a graph G_i that is a directed copy of $G[V_{u_{i-1}} \cup V_{u_i}]$. (Note that G_i is indeed bipartite, since G is H -colored and H contains no self-loops.) Pictorially, the sequence of bipartite graphs is $M_0 G_1 M_1 \dots M_{k-1} G_k M_k$. We also add some additional *gadget edges* between all M_i and M_j with $i \neq j$ and $u_i = u_j$; note that every M_i is

paired with exactly one M_j in this way, because the Eulerian path visits every vertex exactly twice. The gadget edges are the only edges that may be directed from right to left and that connect non-adjacent layers.

The paths p in G that we wish to count modulo two should correspond to the colourful H -subgraphs of G . The path p is supposed to run from left to right through G' ; intuitively, the edge that the path picks at layer M_i corresponds to the vertex of $V_{u_i} \subseteq V(G)$ that $u_i \in V(H)$ is mapped to in the subgraph embedding, and the edge that the path picks at layer G_i corresponds to the edge of G that $u_{i-1}u_i \in E(H)$ is mapped to in the subgraph embedding. The gadget edges ensure that those paths cancel modulo two that do not consistently select the “same” vertex in V_{u_i} and V_{u_j} when $u_i = u_j$.

We now describe the construction of G' in detail.

1. **Graph edges.** For each $i \in \{1, \dots, k\}$, let G_i be a fresh copy of $G[V_{u_{i-1}} \cup V_{u_i}]$, renamed so that $L(G_i) = \{i\} \times V_{u_{i-1}}$ and $R(G_i) = \{i\} \times V_{u_i}$, and directed from left to right.
2. **Matching edges.** For each $i \in \{0, \dots, k\}$, let M_i be the canonical perfect matching between $L(M_i) = \{i\} \times V_{u_i}$ and $R(M_i) = \{i+1\} \times V_{u_i}$, and directed from left to right. Note that $L(M_i) = R(G_i)$ holds for $i \in \{1, \dots, k\}$ and $R(M_i) = L(G_{i+1})$ holds for $i \in \{0, \dots, k-1\}$.
3. **Gadget edges.** For all $i, j \in \{1, \dots, k\}$ with $i < j$ and $u_i = u_j$, note that $L(M_i) = \{i\} \times V_{u_i}$ and $L(M_j) = \{j\} \times V_{u_i}$. We add the canonical *bidirected* perfect matching between $L(M_i)$ and $L(M_j)$. Similarly, we add the canonical bidirected perfect matching between $R(M_i)$ and $R(M_j)$.
4. **Source/sink.** Let s be a new vertex and add all edges (s, v) for $v \in L(M_0)$. Let t be a new vertex and add all edges (v, t) for $v \in R(M_{k+1})$.
5. **Parameters.** Finally, we set $k' = 2k + 2$ and $f(k') = 6k + 3$ so that we are counting all s, t -paths whose length is between k' and $f(k')$.

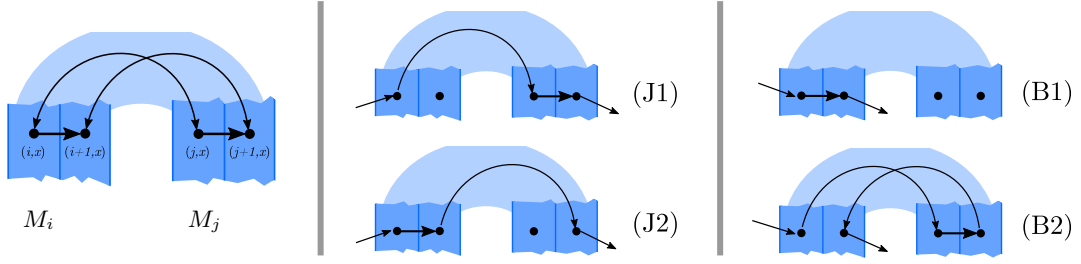
This finishes the construction of G' . Note that $G' \setminus \{s, t\}$ is indeed a sequence $B_0 \dots B_\ell$ of bipartite graphs with some additional gadget edges, where $\ell = 2k$. We define the j -th layer of G' as the set $L_j = L(B_j)$ for $j \leq \ell$ and $L_{\ell+1} = R(B_\ell)$. Recall that $L_j = R(B_{j-1})$ holds for $j > 0$.

We describe the *canonical* solutions in the output of the reduction.

To this end, let H' be an H -subgraph of G that is colourful. This means that H is isomorphic to H' and that the “coloring” homomorphism $c: V(G) \rightarrow V(H)$ is bijective on $V(H')$. Moreover, c restricted to $V(H')$ is in fact an isomorphism: It must map non-edges to non-edges because $E(H)$ and $E(H')$ have the same size. Let $\phi: V(H) \rightarrow V(H')$ be the inverse of c restricted to $V(H')$ and note that ϕ is an isomorphism from H to H' . Moreover, because H has non-trivial automorphisms, this isomorphism ϕ is unique for H' .

We define the *canonical s, t -path spt in G' corresponding to H'* : The path p visits exactly one vertex from each layer from left to right; each layer has the form $L_j = \{i\} \times V_u$ for some i and u , and p chooses the vertex $(i, \phi(u)) \in L_j$ in this layer. Note that this determines all vertices of p . We claim that $V(p)$ indeed induces a path on the graph and matching edges.

To show that p is a path, let $j \in \{0, \dots, \ell\}$. We claim that the two vertices in $V(p) \cap V(B_j)$ are adjacent in B_j . If B_j is one of the matching graphs, then $L(B_j) = \{i\} \times V_{u_i}$ and $R(B_j) = \{i+1\} \times V_{u_i}$ for some i . Since the perfect matching is canonical, there is indeed an edge from $(i, \phi(u_i))$ to $(i+1, \phi(u_i))$ in B_j . Otherwise, B_j is one of the graph copies, say $L(B_j) = \{i\} \times V_{u_{i-1}}$ and $R(B_j) = \{i\} \times V_{u_i}$. Recall that $u_{i-1}u_i$ is part of the Eulerian path and thus an edge of H . Since ϕ is a graph homomorphism from H to G respecting the coloring, we have that $\phi(u_{i-1})\phi(u_i)$ is an edge in $G[V_{u_{i-1}} \cup V_{u_i}]$. Since B_j was a copy of this graph by construction, there is an edge from $(i, \phi(u_{i-1}))$ to $(i, \phi(u_i))$. Overall, we get that spt is an s, t -path in G' , and its length is $\ell + 2 = 2k + 2 = k'$; this is the canonical path corresponding to H' .



■ **Figure 4** The left drawing shows the local configuration around the four vertices of G' that represent a vertex $x \in V_u$ from G . These four vertices are contained in two matchings M_i and M_j for $u_i = u_j = u$. Thick edges are contained in M_i or M_j while light edges are gadget edges. Note that (i, x) and (j, x) have only the depicted outgoing edges in the entire graph G' , and $(i+1, x)$ and $(j+1, x)$ have only the depicted incoming edges. The four drawings on the right depict (up to the symmetry of exchanging i and j) all different ways in which paths might not be canonical: Either they illegally use the gadget edges to jump from M_i to M_j as in (J1) or (J2) and continue from there, or they do not consistently visit the corresponding edges in M_i and M_j , which allows them to use the gadget edges to either take (B2) or not take (B1) a short detour from M_i to M_j .

In summary, every vertex-colorful H -subgraph H' in G defines a unique canonical s, t -path in G' , which implies that the number of canonical paths is equal to the number of H -subgraphs. We now characterize canonical paths slightly differently: Let p be any s, t -path in G' that picks exactly one vertex of each layer from left to right with the additional property that it consistently picks the “same” vertex from each color class. That is, whenever p picks (i, x) in layer $\{i\} \times V_u$ and (j, y) in a layer $\{j\} \times V_u$ (with the same V_u), then $x = y$. Such a path p describes a set of $|V(H)|$ vertices and k edges in G that make up a colorful H -subgraph H' of G , which means that every such path is canonical.

Let \mathcal{P} be the set of all s, t -paths whose length r satisfies $k' \leq r \leq f(k')$. The central claim is that the number of non-canonical paths in \mathcal{P} is even. For this, we construct a fixed-point free involution π on non-canonical paths.

First we focus on paths that are *jumpy* (cf. Figure 4): Let $i, j \in \{0, \dots, k\}$ with $i \neq j$ and $u_i = u_j$. By construction, we added gadget edges between M_i and M_j . Recall that vertices have the form

$$(i, x) \in L(M_i), (i+1, x) \in R(M_i), \quad (j, x) \in L(M_j), (j+1, x) \in R(M_j).$$

A path $p \in \mathcal{P}$ is *jumpy* at i, j, x if

- (J1) p uses the edge from (i, x) to (j, x) but not the edge from $(j+1, x)$ to $(i+1, x)$, or
- (J2) p uses the edge from $(i+1, x)$ to $(j+1, x)$ but not the edge from (j, x) to (i, x) .

If p is jumpy (for some choice of i, j, x), we define $\pi(p)$ as follows: First we identify the lexicographically first position i, j, x where p is jumpy. Then we exchange state (J1) with state (J2) at that position. Note that (J1) implies that p uses the M_j -edge from (j, x) to $(j+1, x)$, since (j, x) has no other outgoing edges, and (J2) implies that p uses the M_i -edge from (i, x) to $(i+1, x)$, because $(i+1, x)$ has no other incoming edges; we swap these edges from M_i and M_j too when applying π . Now π is a fixed-point free involution on jumpy paths, and note that $\pi(p)$ has the same length as p .

Since jumpy paths will cancel out when counting modulo two, we can focus on non-canonical paths that are not jumpy. Paths p that are not jumpy have the following property: A gadget edge from (i, x) on the left side of M_i to (j, x) on the left side of M_j is used by p if and only if the corresponding edge from $(j+1, x)$ to $(i+1, x)$ on the right side is used.

Next we consider paths that are *bad* (again, cf. Figure 4): A path $p \in \mathcal{P}$ is bad at i, j, x with $u_i = u_j$ if

- (B1) p uses the M_i -edge from (i, x) to $(i + 1, x)$ but not the M_j -edge from (j, x) to $(j + 1, x)$,
or
- (B2) p does not use the M_i -edge from (i, x) to $(i + 1, x)$ but does use the M_j -edge from (j, x) to $(j + 1, x)$.

We define $\pi(p)$ for a bad path p by finding the first position i, j, x at which p is bad, and switching between these two states. Say, p was in state (B1) at i, j, x as depicted in the figure, then $\pi(p)$ is in state (B2) at i, j, x , and $\pi(p)$ is exactly two edges longer than p .

We claim that all bad paths that are not jumpy pair up in this manner, without having to consider arbitrarily long paths. Indeed, since p is not jumpy, when we look at the vertices that p traverses, we are for the most part traversing the layers in a monotone order, except for potential short two-vertex detours in bad positions as depicted in the figure as (B2). More precisely, for every vertex v on p , if $v \in L_j$ for $j < \ell$, then either the next vertex is in L_{j+1} or the third vertex after it is in L_{j+1} . This means that the path moves to the right by one layer at least once every 3 vertices, and thus paths that are not jumpy have length at most $3\ell + 3 = 6k + 3 = f(k')$, accounting for $\ell + 1$ matching or graph edges and up to 2ℓ gadget edges that p might take in the short detours, and the two edges at the source and sink.

Finally, if an s, t -path is neither jumpy nor bad, then it does not use any gadget edges, and thus is canonical. Since all jumpy or bad paths cancel, the number of s, t -paths of length between k' and $f(k')$ in G' is the number of canonical paths modulo two. ◀

For completeness, we include two simple reductions: First from the flexible-length to the fixed-length problem in directed graphs, then from the directed to undirected problem.

▶ **Lemma 12.** *Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be computable and strictly increasing. There is a parsimonious polynomial-time fpt-reduction from #Directed f -Flexible Paths to #Directed Paths.*

▶ **Lemma 13.** *#Directed Paths admits a parsimonious poly-time fpt-reduction to #Paths.*

With all these prerequisites collected, we can complete the proof.

Proof of Theorem 3. Let \mathcal{H} be the class of all graphs H that are connected, almost 4-regular, and whose automorphism group has size one. We use the probabilistic method to argue that the tree-width of graphs in \mathcal{H} is not bounded. With probability $1 - o(1)$ as $h \rightarrow \infty$, random 4-regular graphs with h vertices are connected [41, Theorem 2.10], they have no nontrivial automorphisms [27], and they are almost Ramanujan [24, Theorem 7.10], that is, their second-largest eigenvalue in absolute value satisfies $\lambda \leq 2\sqrt{3} + o(1) < 3.5$. By a union bound, H has all three properties simultaneously with probability $1 - o(1)$. By Cheeger's inequality [24, Theorem 4.11], we have

$$\min_{S \subseteq V(H), |S| \leq \frac{1}{2}h} \frac{|E(S, \bar{S})|}{|S|} \geq \frac{1}{2}(4 - \lambda) > 0.1,$$

that is, the edge expansion is bounded away from zero, which implies that the tree-width of H is at least linear in h (see, e.g., [12, Exercise 7.34]). Now, if we remove an arbitrary edge e from H , we obtain an almost 4-regular graph that remains connected (since H is Eulerian) and whose tree-width has decreased by at most 1. Moreover, suppose that π is an automorphism of $H - e$. Since π preserves degrees, it has to map the vertex set e to e . But then π is an automorphism of H , too, which implies that π is the trivial automorphism and the automorphism group of $H - e$ has size one as required. Thus, \mathcal{H} contains graphs of arbitrarily large tree-width.

By Lemma 5, $\oplus\text{VertexColorfulSub}(\mathcal{H})$ is $\oplus\text{W}[1]$ -hard under parsimonious fpt-reductions. If there is a parsimonious fpt-reduction from problem $\#A$ to problem $\#B$ then in particular the parity version $\oplus A$ reduces to $\oplus B$. Writing $\oplus A \leq \oplus B$ we can summarize the chain of reductions in Lemmas 11–13 as

$$\oplus\text{VertexColorfulSub}(\mathcal{H}) \leq \oplus\text{Directed } f\text{-Flexible Paths} \leq \oplus\text{Directed Paths} \leq \oplus\text{Paths}.$$

This proves the $\oplus\text{W}[1]$ -hardness of $\oplus\text{Paths}$. The containment follows from the standard fpt-reduction from $\#Paths$ to $\#Clique$, which is parsimonious. Overall, the claim follows. \blacktriangleleft

6 Conclusion

We conducted an initial investigation of modular subgraph counting, leading to the partial classification depicted in Figure 1. To obtain a complete picture, the following conjecture needs to be addressed.

► **Conjecture 14.** *For any computable pattern class \mathcal{H} :*

- *If \mathcal{H} has unbounded matching-split number, then the problem $\oplus\text{Sub}(\mathcal{H})$ is $\oplus\text{W}[1]$ -complete.*
- *If \mathcal{H} has unbounded vertex-cover number, then $\#\text{Sub}(\mathcal{H}) \bmod q$ for fixed $q \in \mathbb{N}$ is $\text{Mod}_p\text{W}[1]$ -complete for any odd divisor p of q .*

An appropriate transfer of the subgraph-homomorphism framework to modular counting is likely to help in settling this conjecture. Partial results towards this have been obtained by Peyerimhoff et al. [33].

References

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- 2 Vikraman Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In Prosenjit Bose and Pat Morin, editors, *Algorithms and Computation, 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21–23, 2002, Proceedings*, volume 2518 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2002. doi:10.1007/3-540-36136-7_40.
- 3 Andreas Björklund, Holger Dell, and Thore Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2015. doi:10.1007/978-3-662-47672-7_19.
- 4 Andreas Björklund and Thore Husfeldt. The parity of directed Hamiltonian cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26–29 October, 2013, Berkeley, CA, USA*, pages 727–735. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.83.
- 5 Hans L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, January 1993. doi:10.1006/jagm.1993.1001.
- 6 Yijia Chen, Martin Grohe, and Bingkai Lin. The hardness of embedding grids and walls. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21–23, 2017, Revised Selected Papers*, volume 10520 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2017. doi:10.1007/978-3-319-68705-6_14.

- 7 Radu Curticapean. *The simple, little and slow things count: on parameterized counting complexity*. PhD thesis, Saarland University, 2015. URL: <http://scidok.sulb.uni-saarland.de/volltexte/2015/6217/>.
- 8 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. 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 210–223. ACM, 2017. doi:10.1145/3055399.3055502.
- 9 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.22.
- 10 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. *CoRR*, abs/1407.2929, 2014. arXiv:1407.2929.
- 11 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Genus, apices, minors, evaluation mod 2k. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 994–1009. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.65.
- 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 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.
- 14 Julian Dörfler, Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting induced subgraphs: An algebraic approach to $\#W[1]$ -hardness. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 26:1–26:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.26.
- 15 John D. Faben. The complexity of counting solutions to generalised satisfiability problems modulo k. *CoRR*, abs/0809.1836, 2008. arXiv:0809.1836.
- 16 John D. Faben and Mark Jerrum. The complexity of parity graph homomorphism: An initial investigation. *Theory Comput.*, 11:35–57, 2015. doi:10.4086/toc.2015.v011a002.
- 17 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.
- 18 Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Živný. Counting homomorphisms to K_4 -minor-free graphs, modulo 2. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2303–2314. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.137.
- 19 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012. doi:10.1016/j.jcss.2011.10.001.
- 20 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. The complexity of counting homomorphisms to cactus graphs modulo 2. *ACM Trans. Comput. Theory*, 6(4):17:1–17:29, 2014. doi:10.1145/2635825.
- 21 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. Counting homomorphisms to square-free graphs, modulo 2. *ACM Trans. Comput. Theory*, 8(3):12:1–12:29, 2016. doi:10.1145/2898441.
- 22 Heng Guo, Sangxia Huang, Pinyan Lu, and Mingji Xia. The complexity of weighted Boolean $\#CSP$ modulo k. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011*,


- Dortmund, Germany, volume 9 of *LIPICs*, pages 249–260. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICs.STACS.2011.249.
- 23 Hiroshi Hirai and Hiroyuki Namba. Shortest $(a+b)$ -path packing via Hafnian. *Algorithmica*, 80(8):2478–2491, 2018. doi:10.1007/s00453-017-0334-0.
 - 24 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(04):439–562, August 2006. doi:10.1090/s0273-0979-06-01126-8.
 - 25 Bart M. P. Jansen and Dániel Marx. Characterizing the easy-to-find subgraphs from the viewpoint of polynomial-time algorithms, kernels, and Turing kernels. 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 616–629. SIAM, 2015. doi:10.1137/1.9781611973730.42.
 - 26 Amirhossein Kazeminia and Andrei A. Bulatov. Counting homomorphisms modulo a prime number. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 59:1–59:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.59.
 - 27 Jeong Han Kim, Benny Sudakov, and Van H. Vu. On the asymmetry of random regular graphs and random graphs. *Random Struct. Algorithms*, 21(3-4):216–224, 2002. doi:10.1002/rsa.10054.
 - 28 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: Track 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.
 - 29 Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discret. Math.*, 27(2):892–909, 2013. doi:10.1137/110859798.
 - 30 Bingkai Lin. The parameterized complexity of the k -biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.
 - 31 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
 - 32 Burkhard Monien. How to find long paths efficiently. In *Analysis and Design of Algorithms for Combinatorial Problems*, pages 239–254. Elsevier, 1985. doi:10.1016/s0304-0208(08)73110-4.
 - 33 Norbert Peyerimhoff, Marc Roth, Johannes Schmitt, Jakob Stix, and Alina Vdovina. Parameterized (modular) counting and Cayley graph expanders. *CoRR*, abs/2104.14596, 2021. arXiv:2104.14596.
 - 34 Jürgen Plehn and Bernd Voigt. Finding minimally weighted subgraphs. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 16th International Workshop, WG '90, Berlin, Germany, June 20-22, 1990, Proceedings*, volume 484 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 1990. doi:10.1007/3-540-53832-1_28.
 - 35 Marc Roth. Counting restricted homomorphisms via Möbius inversion over matroid lattices. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPICs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.63.
 - 36 Marc Roth and Johannes Schmitt. Counting induced subgraphs: A topological approach to $\#W[1]$ -hardness. *Algorithmica*, 82(8):2267–2291, 2020. doi:10.1007/s00453-020-00676-9.
 - 37 Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting small induced subgraphs satisfying monotone properties. In *61st IEEE Annual Symposium on Foundations of Computer*

- Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1356–1367. IEEE, 2020. doi:10.1109/FOCS46700.2020.00128.
- 38 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.
- 39 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. 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 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.
- 40 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.
- 41 Nicholas C. Wormald. Models of random regular graphs. *London Mathematical Society Lecture Note Series*, pages 239–298, 1999.

Minimum Common String Partition: Exact Algorithms

Marek Cygan ✉ 🏠 

University of Warsaw, Poland

Alexander S. Kulikov ✉ 🏠 

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia
St. Petersburg State University, Russia

Ivan Mihajlin ✉

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia

Maksim Nikolaev ✉ 

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia

Grigory Reznikov ✉ 

National Research University Higher School of Economics, St. Petersburg, Russia

Abstract

In the minimum common string partition problem (MCSP), one gets two strings and is asked to find the minimum number of cuts in the first string such that the second string can be obtained by rearranging the resulting pieces. It is a difficult algorithmic problem having applications in computational biology, text processing, and data compression. MCSP has been studied extensively from various algorithmic angles: there are many papers studying approximation, heuristic, and parameterized algorithms. At the same time, almost nothing is known about its exact complexity. In this paper, we present new results in this direction. We improve the known 2^n upper bound (where n is the length of input strings) to ϕ^n where $\phi \approx 1.618...$ is the golden ratio. The algorithm uses Fibonacci numbers to encode subsets as monomials of a certain implicit polynomial and extracts one of its coefficients using the fast Fourier transform. Then, we show that the case of constant size alphabet can be solved in subexponential time $2^{O(n \log \log n / \log n)}$ by a hybrid strategy: enumerate all long pieces and use dynamic programming over histograms of short pieces. Finally, we prove almost matching lower bounds assuming the Exponential Time Hypothesis.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Algorithm design techniques

Keywords and phrases similarity measure, string distance, exact algorithms, upper bounds, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.35

Funding *Marek Cygan*: The work of Marek Cygan is part of a project TOTAL that has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation programme (grant agreement No 677651).

Alexander S. Kulikov: The research presented in Section 6 is supported by Russian Science Foundation (18-71-10042).

Maksim Nikolaev: The research presented in Section 6 is supported by Russian Science Foundation (18-71-10042). The research presented in Section 3 is supported by Ministry of Science and Education grant 075-15-2019-1620.



© Marek Cygan, Alexander S. Kulikov, Ivan Mihajlin, Maksim Nikolaev, and Grigory Reznikov; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 35; pp. 35:1–35:16

Leibniz International Proceedings in Informatics



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

1 Overview

Two strings $s_1, s_2 \in \Sigma^n$ are said to have a common partition of size t if one can cut s_1 into t blocks (by $t-1$ cuts), rearrange them, and get s_2 . This is a string similarity measure having applications in computational biology, text processing, and data compression. The minimum common string partition problem (MCSP) is to find the minimum size of a common partition of two input strings. There are two natural special cases of the problem: in d -MCSP, every symbol appears at most d times in each input string; in MCSP^c, the size of the alphabet Σ is at most c .

1.1 Known results

The problem has been studied extensively from various algorithmic angles.

Computational complexity. Already 2-MCSP is APX-hard, hence MCSP is both NP-hard and APX-hard [8].

Parameterized algorithms. The problem is fixed parameter tractable (FPT) with respect to combinations of various parameters [6, 11, 2, 3]. For example, [2] gives an $O^*(d^{2k})$ algorithm¹ for d -MCSP, whereas [3] gives an FPT algorithm with respect to t only.

Approximation algorithms. The best known approximation ratio for a polynomial-time algorithm is $O(\log n \log^* n)$ [5]. There is also an $O(d)$ -approximation algorithm [15]. The best known approximation ratios for 2-MCSP and 3-MCSP are 1.1037 and 4, respectively [8].

Heuristic algorithms. One natural heuristic approach for MCSP is greedy: cut out a longest common substring and repeat. It is well studied both from practical and theoretical points of view. Its approximation factor is: $O(\log n)$ for many families of inputs [17], between $\Omega(n^{0.46})$ and $O(n^{0.69})$ in the worst case [4, 13]. It can be implemented in linear time [9].

1.2 New results

Much less is known about *the exact* complexity of MCSP. The best known upper bound is $O^*(2^n)$ [7]. This aligns well with what is known for many other permutation problems: say, the shortest common superstring problem, the shortest common supersequence problem, the traveling salesman problem. Whereas many approximation, parameterized, and heuristic algorithms are known for these problems, breaking the 2^n barrier for any of them is a long standing open problem. (For the mentioned problems, n denotes the number of input strings, sequences, or nodes.)

In this paper, we present the following new exact algorithms and lower bounds for MCSP. We start by showing two straightforward $O^*(2^n)$ algorithms: the first one enumerates all possible partitions of both input strings and is perhaps the simplest exact algorithm for the problem; the second one is a straightforward application of the dynamic programming method. Both algorithms use exponential space.

¹ The $O^*(\cdot)$ notation is common in the field of algorithms for NP-hard problems: like $O(\cdot)$ hides constant factors, it suppresses factors that grow polynomially in the input length.

1.2.1 Improving time to ϕ^n : dynamic programming

Then, we show how to improve the dynamic programming algorithm to get the running time $O^*(\phi^n)$ where $\phi \approx 1.618\dots$ is the golden ratio. In short, the improvement is based on the following idea. For two strings to have a common partition, their multisets of symbols should coincide. Then, if one cuts the first string into blocks and maps all blocks of length at least two to the second string (so that the mapped parts do not overlap), all the remaining blocks of length one are mapped automatically.

1.2.2 Improving space to polynomial: FFT and Fibonacci encoding

The improved algorithm still uses exponential space (as it is based on dynamic programming). Again, avoiding exponential space in algorithms for NP-hard problems may be a challenging task. For example, the best known algorithm for the coloring problem has running time $O^*(2^n)$ and uses exponential space [1]; bringing down the space consumption to polynomial is an open problem. Also, for the optimal 2-constraint satisfaction problem (as well as for its special cases: maximum 2-satisfiability and maximum cut), all known better-than- 2^n -time algorithms use exponential space [18, 14]; improving 2^n time while keeping polynomial space is an open problem. For the MCSP problem, we show how to get a polynomial space algorithm with running time $O^*(\phi^n)$.

The approach that we use is inspired by generating functions. Let $c_0, c_1, c_2, \dots = \{c_i\}_{i=0}^\infty$ be an integer sequence of interest, where term c_i is expressed in a complex way through the previous terms, so that it is difficult to directly find a closed form for it. Occasionally, one can solve this problem in three steps.

1. Pack a sequence into a polynomial (or, rather, formal power series):

$$T(x) = c_0 + c_1x + c_2x^2 + \dots = \sum_{i=0}^{\infty} c_i x^i.$$

2. By means of algebraic manipulations, find a closed form expression for $T(x)$.
3. Then, c_i is the coefficient of x_i in Taylor expansion of $T(x)$ that can be found using the value of the i^{th} derivative of T at zero.

What we use can be viewed as a discrete and algorithmic version of this approach. Given strings $s_1, s_2 \in \Sigma^n$, we define a finite sequence $\{c_i\}_{i=0}^N$ with the following property: s_1, s_2 have a common partition of size t iff a particular term c is positive. This term may be expressed through the other terms, but it will take a lot of space, so instead we find it in three steps.

1. Pack a sequence into a polynomial P of finite degree such that the term c is a coefficient of a particular monomial m in P .
2. Describe an efficient way of evaluating P at a given point.
3. Using fast Fourier transform, extract the coefficient of m using values of P at specific points.

The algorithm uses Fibonacci encoding in order to keep the degree of the polynomial low enough.

1.2.3 Subexponential running time for small alphabets: hybrid strategy

Then, we show that for the constant size alphabets, one can even get a subexponential running time: $2^{O(\frac{n \log \log n}{\log n})}$. The main idea is to track short and long blocks differently: we just guess the cut points of all long blocks (this is feasible as there are not too many of them) and we exploit dynamic programming to handle the short ones. By balancing these two cases carefully, one gets a speed up to a subexponential time.

1.2.4 Lower bounds

Finally, we show that substantial improvement of the presented upper bounds is difficult. Namely, we prove that under the Exponential Time Hypothesis (stating that there are no subexponential time algorithms for the satisfiability problem), MCSP cannot be solved in subexponential time (of the form $2^{o(n)}$). This is a simple application of the NP-hardness proof of MCSP. Then, we present a reduction from the general case to the special case of constant size alphabet that gives a lower bound $2^{\Omega(n/\log n)}$ (under ETH) for that special case.

Notation

Throughout the paper, we use the following notation. Let s be a string of length n . We use 1-based indexing and slice notation: $s = s[1]s[2] \cdots s[n]$; for $1 \leq l \leq r \leq n$, $s[l:r] = s[l]s[l+1] \cdots s[r]$. By $[n]$ we denote the set $\{1, \dots, n\}$.

2 2^n time and space: enumerating all partitions

In this section, we present perhaps the most straightforward algorithm. As simple as it is, it has an important feature that many other algorithms lack: it works for any number of input strings. See Algorithm 1. Its running time is $O^*(2^n)$ as the number of different partitions of a string of length n is 2^{n-1} : there are $n-1$ places to make a cut.

■ **Algorithm 1** Enumerating all partitions.

Input: strings s_1, \dots, s_j of length n .
Output: the minimum size of a common partition of s_1, \dots, s_j .

```

1:  $T \leftarrow$  associative array
2: for every partition of  $s_1$  into substrings  $p_1, \dots, p_k$  do
3:    $T[\text{multiset}\{p_1, \dots, p_k\}] \leftarrow j\text{-tuple } (1, 0, \dots, 0)$ 
4: for  $i$  from 2 to  $j$  do
5:   for every partition of  $s_i$  into substrings  $p_1, \dots, p_k$  do
6:     if  $T$  has key  $\text{multiset}\{p_1, \dots, p_k\}$  then
7:        $T[\text{multiset}\{p_1, \dots, p_k\}][i] \leftarrow 1$ 
8:  $t \leftarrow +\infty$ 
9: for every key  $P$  of  $T$  do:
10:  if  $T[P] = (1, \dots, 1)$  then
11:     $t \leftarrow \min(t, |P|)$ 
12: return  $t$ 

```

Hereinafter, we return the size of the optimal partition instead of the partition itself. This is equivalent to the original problem, since there is a polynomial Algorithm 2 for finding the optimal partition given an oracle for minimum size of a common partition. In this algorithm, we first look for the longest prefix p of s_1 that can be matched somewhere in s_2 , such that after replacing it and its match with a new symbol $\#_1$ the size of the optimal partition stay the same. To make sure that the size of the optimal partition does not change, we use the oracle. It is clear that in the optimal partition $\#_1$ is not a prefix of some longer block. After we find p , we replace it and its match with $\#_1$ and start looking for the next longest block in the optimal partition that we replace with a new symbol $\#_2$ and so on. In the end we obtain a partition $\mathcal{P} = \{p_1, p_2, \dots, p_t\}$ such that $s_1 = p_1 \dots p_t$ and p_i is the longest possible block in a common partition of the minimum size t given previous $i-1$ blocks.

■ **Algorithm 2** Constructing the optimal partition.

Input: strings s_1, s_2 of length n , oracle \mathcal{O} for the minimum size of a common partition.
Output: a common partition of s_1, s_2 of the minimum size.

```

1:  $\mathcal{P} \leftarrow \emptyset$ 
2:  $t \leftarrow \mathcal{O}(s_1, s_2)$ 
3: for  $i$  from 1 to  $t$  do
4:    $\#_i \leftarrow$  a symbol that does not occur in  $s_1$  and  $s_2$ 
5:   for all  $j$  from  $n$  down to  $i$ , all  $1 \leq l \leq r \leq n$  do
6:      $s'_1 \leftarrow s_1[1 : i - 1] \#_i s_1[j + 1 : n]$ 
7:      $s'_2 \leftarrow s_2[1 : l - 1] \#_i s_2[r + 1 : n]$ 
8:     if  $s_1[i : j] = s_2[l : r]$  and  $\mathcal{O}(s'_1, s'_2) = t$  then
9:        $\mathcal{P} \leftarrow \mathcal{P} \cup s_1[i : j]$ 
10:     $s_1 \leftarrow s'_1$ 
11:     $s_2 \leftarrow s'_2$ 
12:    break
13: return  $\mathcal{P}$ 

```

3 Improving time to ϕ^n : dynamic programming

Here, we present a dynamic programming solution with roughly the same running time and space as the previous solution. Later, we will be able to improve both time and space of this algorithm. The algorithm works by solving the following subproblem: for $0 \leq k \leq n$, let $C(k)$ be the set of all pairs (S, t) , where $S \subseteq [n]$, $|S| = k$, and $1 \leq t \leq n$, such that one can cut $s_1[1 : k]$ into t blocks and match them to the subsequence of s_2 specified by S . The minimum size of a common partition is then simply the minimum t such that (S, t) is contained in $C(n)$. To compute $C(k)$, we first fix the length i of the last block of $s_1[1 : k]$. Then, this block should be matched somewhere in s_2 , whereas $C(k - i)$ can be used to find the positions where the remaining $t - 1$ blocks should be matched. The formal pseudocode is given in Algorithm 3. The running time and space is $O^*(2^n)$ as the number of different subsets is 2^n whereas all other steps clearly take polynomial time.

■ **Algorithm 3** Dynamic programming.

Input: strings s_1, s_2 of length n .
Output: the minimum size of a common partition of s_1, s_2 .

```

1:  $C(0) \leftarrow \{(\emptyset, 0)\}$ 
2: for  $k$  from 1 to  $n$  do
3:    $C(k) \leftarrow \emptyset$ 
4:   for all  $1 \leq i \leq k$ , all  $1 \leq l \leq n - i + 1$ , all  $(S, t) \in C(k - i)$  do
5:     if  $s_1[k - i + 1 : k] = s_2[l : l + i - 1]$  and  $S \cap \{l, l + 1, \dots, l + i - 1\} = \emptyset$  then
6:        $C(k) \leftarrow C(k) \cup \{(S \cup \{l, l + 1, \dots, l + i - 1\}, t + 1)\}$ 
7: return  $\min\{t : (S, t) \in C(n)\}$ 

```

In the previous algorithm, we considered subsets of s_2 that have a common partition of particular size with the prefix of s_1 . Here, we are going to ignore pieces of size one in every such partition: indeed, if one successfully matched longer pieces, then the pieces of size one will match as well.

With this in mind, one may consider only subsets of s_2 that can be cut into pieces of length at least two and matched to some subset of the prefix of s_1 : the remaining part of the prefix is assumed to be filled with pieces of length one.

In addition to ignoring pieces of length one, we also plug the “holes” of size one in our subsets, since we can plug them only with pieces of size one and may do it immediately. To formalize that, we introduce the function Plug whose pseudocode is given in Algorithm 4. Say that there is a hole at position $i \in [n]$ in a set $S \subseteq [n]$ if $i \notin S$ whereas each of $i - 1$ and $i + 1$ either belong to S or does not belong to $[n]$.

■ **Algorithm 4** The function Plug.

Input: subset $S \subseteq [n]$.
Output: hole-free superset of S of minimum size.

```

1: for  $i$  in  $[n]$  do
2:   if  $S$  has a hole at position  $i$  then
3:      $S \leftarrow S \cup \{i\}$ 
4: return  $S$ 

```

Now we are ready to describe the subproblem of our algorithm. For $0 \leq k \leq n$, let $C(k)$ be the set of all pairs (S, t) , for which there exist a pair (T, β) such that $S = \text{Plug}(T)$, $t = \beta + |S| - |T|$, and one can cut the subsequence of s_2 specified by T into β blocks of size at least two and match them to some subsequence of $s_1[1 : k]$. Thus, t is a number of blocks into which S can be cut so that all the blocks of size at least two match to $s_1[1 : k]$ and blocks of size one (that is, plugged holes in T) match somewhere in s_1 (not necessarily in $s_1[1 : k]$). The minimum size of a common partition is then the minimum value of $t + n - |S|$ such that (S, t) is contained in $C(n)$. Here, $n - |S|$ is the number of ignored but not plugged blocks of size one. Clearly, $C(k) \subseteq C(k')$ for $k < k'$. To compute $C(k)$, we first fix the length $i \neq 1$ of the long block $s_1[k - i + 1 : k]$ and $j \geq 0$ blocks of size one before it that we wish to ignore. Then, this long block should be matched somewhere in s_2 , whereas $C(k - i - j)$ can be used to find the positions where the remaining long blocks should be matched. After we correctly match the long block somewhere in s_2 , we append it to the corresponding S from $C(k - i - j)$, plug the holes and recalculate the number of blocks as the number of blocks in S before appending plus one plus the number of holes plugged after appending. If $i = 0$, then we may skip the choice of j and simply get $C(k - 1)$. The formal pseudocode is given in Algorithm 5.

■ **Algorithm 5** Improved dynamic programming algorithm.

Input: strings s_1, s_2 of length n .
Output: the minimum size of a common partition of s_1, s_2 .

```

1:  $C(0) \leftarrow \{(\emptyset, 0)\}$ 
2: for  $k$  from 1 to  $n$  do
3:    $C(k) \leftarrow C(k - 1)$  (case  $i = 0$ )
4:   for all  $2 \leq i \leq k$ , all  $0 \leq j \leq n - i$ , all  $1 \leq l \leq n - i + 1$ , all  $(S, t) \in C(k - i - j)$  do
5:     if  $s_1[k - i + 1 : k] = s_2[l : l + i - 1]$  and  $S \cap \{l, l + 1, \dots, l + i - 1\} = \emptyset$  then
6:        $S' \leftarrow \text{Plug}(S \cup \{l, l + 1, \dots, l + i - 1\})$ 
7:        $C(k) \leftarrow C(k) \cup \{(S', t + 1 + |S'| - |S|)\}$ 
8: return  $\min\{t + n - |S| : (S, t) \in C(n)\}$ 

```

► **Theorem 1.** Algorithm 5 solves MCSP in time and space $O^*(\phi^n)$.

Proof. Correctness. For every $(S, t) \in C(n)$, the total number of blocks in the corresponding partition is the number of long blocks plus the number of plugged holes ($= t$) plus $|[n] \setminus S|$ blocks of size one we ignore. If the optimal partition contains no pieces longer than one, then we never fulfill the if condition, but nevertheless $C(n)$ will contain $(\emptyset, 0)$ dragged through the whole cycle (thanks to line 3), that corresponds to the partition of size n as required.

Running time. It is sufficient to count all the subsets S we considered. For that, it is convenient to treat every such subset as a sequence $v \in \{0, 1\}^n$ without lonely zeros and ones. Let $f_0(n)$ be the number of such sequences that end up with zero and $f_1(n)$ be the number of such sequences that end up with one. Since every such sequence ending with one ends with at least two ones, we can write the recurrence relation $f_1(n) = \sum_{i=2}^{n-2} f_0(i)$. Similarly, $f_1(n) = f_0(n)$, hence $f_1(n) = \sum_{i=2}^{n-2} f_1(i)$ that, together with the initial conditions $f_1(2) = f_1(3) = 1$, describes the shifted sequence of Fibonacci numbers, thus $f_1(n) = O(\phi^n)$. ◀

4 Improving space to polynomial: FFT and Fibonacci encoding

In this section, we improve the space of the previously considered algorithm to polynomial. The following theorem provides a basic toolkit for this.

► **Theorem 2.** Assume that for two strings $s_1, s_2 \in \Sigma^n$ and a parameter t one can construct a multivariate polynomial (of finite degree) $P(X)$ over a (finite) variable set X and a monomial $m = \prod_{x_i \in X} x_i^{d_i}$, where $d_i \in \mathbb{Z}_{\geq 0}$, with the following three properties:

1. Its coefficients are non negative and less than $W = 2^{\text{poly}(n)}$.
2. For fixed values of variables X , one can compute $P(X)$ in time polynomial in the length of the binary representation of the values.
3. There is a common partition of s_1, s_2 of size t if and only if the coefficient of m in P is non-zero.

Then, MCSP can be solved in time

$$O^* \left(\text{polylog} \left(\prod_{x_i \in X} \deg_{x_i}(P) \right) \prod_{x_i \in X} \deg_{x_i}(P) \right)$$

and space

$$O^* \left(\text{polylog} \left(\prod_{x_i \in X} \deg_{x_i}(P) \right) \right),$$

where $\deg_{x_i}(P)$ is the degree of variable x_i in P , that is, the maximum degree of x_i across the monomials of P with non-zero coefficients.

In the proof, we use the following theorem proved in [10] (Theorem 3.2).

► **Theorem 3.** Let $P(x) = \sum_{i=0}^d p_i x^i$ be a polynomial of degree at most d with non-negative integer coefficients less than W . Given an arithmetic circuit $C(x, p)$ of size $\text{polylog}(d, W)$ which evaluates P modulo a prime $p = d \text{polylog}(d, W)$ at an integer point x , any coefficient of $P(x)$ can be found in time $d \text{polylog}(d, W)$ and space $\text{polylog}(d, W)$.

Here we calculate modulo p just to prevent the numbers we operate from growing exponentially fast.

Proof of Theorem 2. Let $X = \{x_1, \dots, x_q\}$. In order to apply Theorem 3, we need to build a univariate polynomial $Q(x)$ out of $P(x_1, \dots, x_q)$. To do this, we use Kronecker substitution [16]:

$$Q(x) = P\left(x, x^{\deg_{x_1}(P)+1}, x^{(\deg_{x_1}(P)+1)(\deg_{x_2}(P)+1)}, \dots, x^{\prod_{i=1}^{q-1}(\deg_{x_i}(P)+1)}\right).$$

That is, we replace x_i by $x^{\prod_{j=1}^{i-1}(\deg_{x_j}(P)+1)}$. Then, P contains a monomial $m = \prod_{i \in [q]} x_i^{a_i}$ if and only if Q contains a monomial $x^{\sum_{i \in [q]} a_i \cdot \prod_{j=1}^{i-1}(\deg_{x_j}(P)+1)}$.

Since $W = 2^{\text{poly}(n)}$, $\text{polylog}(d, W) = O^*(\text{polylog}(d))$. \blacktriangleleft

As a warm up, we show how to reduce the space complexity of the 2^n dynamic programming Algorithm 3 to polynomial.

► **Theorem 4.** MCSP can be solved in time $O^*(2^n)$ and polynomial space.

Proof. We construct a series of polynomials P_k for $0 \leq k \leq n$ associated with the steps of the dynamic programming algorithm. Each P_k corresponds to $C(k)$ in the mentioned algorithm in the following way: P_k has a monomial $\alpha z^t y^{|S|} \prod_{i \in S} x^{2^{i-1}}$ for some $\alpha > 0$ depending on the monomial if and only if $(S, t) \in C(k)$. The pseudocode for computing P_n is given in Algorithm 6. It is not difficult to see that this is a polynomial time algorithm.

■ **Algorithm 6** Computing P_n .

Input: strings s_1, s_2 of length n , values x, y, z .
Output: value of P_n at the point (x, y, z) .

- 1: $P_0(x, y, z) \leftarrow 1$
- 2: $P_k(x, y, z) \leftarrow 0$ for all $1 \leq k \leq n$
- 3: **for** k **from** 1 **to** n **do**
- 4: **for all** $1 \leq i \leq k$, **all** $1 \leq l \leq n - i + 1$ **do**
- 5: **if** $s_1[k - i + 1 : k] = s_2[l : l + i - 1]$ **then**
- 6: $P_k(x, y, z) \leftarrow P_k(x, y, z) + zy^i x^{2^{l+i-1} - 2^{l-1}} P_{k-i}(x, y, z)$
- 7: **return** $P_n(x, y, z)$

We claim that s_1 and s_2 have a common partition of size t if and only if P_n contains a monomial $z^t y^n x^{2^n - 1}$. One direction is straightforward. Assume that there is a partition of s_1 into pieces $(l_1, r_1), \dots, (l_t, r_t)$, where $l_1 = 1, r_t = n$ and $r_i = l_{i+1} - 1$ for all $i \in [t - 1]$ and it corresponds to a partition of s_2 into pieces $(l'_1, r'_1), \dots, (l'_t, r'_t)$ such that $s_1[l_i : r_i] = s_2[l'_i : r'_i]$ for all $i \in [t]$. Then P_{r_i} has a monomial

$$zy^{r_i - l_i + 1} x^{2^{r'_i} - 2^{l'_i - 1}} P_{r_i - 1}.$$

Then, by induction, P_{r_i} has a monomial

$$z^i y^{\sum_{j=1}^i (r_j - l_j + 1)} x^{\sum_{j=1}^i 2^{r'_j} - 2^{l'_j - 1}}.$$

Hence, $P_n = P_{r_t}$ has a monomial

$$z^t y^{\sum_{j=1}^t (r_j - l_j + 1)} x^{\sum_{j=1}^t 2^{r'_j} - 2^{l'_j - 1}} = z^t y^n x^{2^n - 1}.$$

For the reverse direction, suppose P_n has a monomial $z^t y^n x^{2^n-1}$. It could only be obtained as a product $\prod_{i=1}^t z y^{r_i-l_i+1} x^{2^{r_i}-2^{l_i-1}}$ for some $(l_1, r_1), \dots, (l_t, r_t)$ during the execution of the algorithm. An important invariant of each such terms is that the degree of y is equal to the Hamming weight of the degree of x . (Here, by the Hamming weight of an integer we mean the sum of the bits of its binary representation.)

As $\sum_{i=1}^t (r_i - l_i + 1)$ is equal to the degree of y , it is equal to n . If they are all disjoint, we have a valid partition. Suppose there exist i and j such that (l_i, r_i) and (l_j, r_j) intersect. Consider the product

$$(zy^{r_i-l_i+1}x^{2^{r_i}-2^{l_i-1}})(zy^{r_j-l_j+1}x^{2^{r_j}-2^{l_j-1}}) = z^2 y^{r_i+r_j-l_i-l_j+2} x^{\left(\sum_{t=l_i}^{r_i} 2^{t-1}\right) + \left(\sum_{t=l_j}^{r_j} 2^{t-1}\right)}.$$

Now, let us look at the Hamming weight of the degree of x . As it is the sum of $d = r_i + r_j - l_i - l_j + 2$ powers of 2, it is at most d . But as (l_i, r_i) and (l_j, r_j) intersect, there is at least one carry, so it is actually less than d . Then, when we multiply all the terms, as the Hamming weight of the sum is not more than the sum of Hamming weights, we have that Hamming weight is strictly less than the degree of y , so it is less than n . But the Hamming weight of $2^n - 1$ is equal to n , a contradiction.

Finally, we can solve MCSP by finding the smallest t such that $z^t y^n x^{2^n-1}$ is present in P_n . As we already have a polynomial time algorithm for P_n , its coefficients are not greater than $(n^2)^n = 2^{O(n \log n)}$ and total degree is $O^*(2^n)$, by Theorem 2 there exists an $O^*(2^n)$ time and polynomial space algorithm for MCSP. ◀

We are now going to infuse the previous algorithm with ideas from the improved dynamic programming and introduce a better encoding to get a speed up. In the previous section, we were encoding each segment (l, r) as $2^r - 2^{l-1}$. This is a natural way to represent a subset: this number has ones in its binary representation exactly at positions from l to r . Then we were using a property of the binary representation that the sum of encodings of two intersecting segments has a Hamming weight strictly less than sum of the Hamming weights of the terms. Another way to look at it: for any two intersecting segments $(l_1, r_1), (l_2, r_2)$ there is a collection of nonintersecting segments $(l'_1, r'_1), \dots, (l'_p, r'_p)$ such that:

$$\sum_{i=1}^2 2^{r_i} - 2^{l_i-1} = \sum_{i=1}^p 2^{r'_i} - 2^{l'_i-1}, \text{ but}$$

$$\sum_{i=1}^2 r_i - l_i + 1 > \sum_{i=1}^p r'_i - l'_i + 1$$

Here we can take as (l'_i, r'_i) all the substrings of consecutive 1s in the binary representation of $\sum_{i=1}^2 2^{r_i} - 2^{l_i-1}$.

We are going to do something similar but sacrifice some of this clarity for efficiency. We will encode each segment (l, r) as $F'(r) - F'(l-1)$, where $F'(i)$ is the $(i+1)$ -th Fibonacci number (so $F'(-1) = 0$ and $F'(0) = 1$). Then we will show that this encoding has a similar property as a binary one but only for segments of size greater than one.

► **Theorem 5.** MCSP can be solved in time $O^*(\phi^n)$ and polynomial space.

Proof. Consider a polynomial defined by Algorithm 7.

We call a piece of the partition *long* if it has length greater than one. The meaning of the indices is the following:

- k is the length of the prefix of s_1 that we are currently processing,

■ **Algorithm 7** Computing P_n

Input: strings s_1, s_2 of length n , values x, y, z, w .
Output: value of P_n at the point (x, y, z, w) .

- 1: $P_k \leftarrow 1$ for all $0 \leq k \leq n$
- 2: **for** k **from** 1 **to** n **do**
- 3: **for all** $1 < i \leq j \leq k$, **all** $1 \leq l \leq n - i + 1$, **all** $n - i + 1 \leq q \leq n$ **do**
- 4: **if** $s_1[j - i + 1 : j] = s_2[l : l + i - 1]$ **then**
- 5: $r \leftarrow l + i - 1$
- 6: $P_k \leftarrow P_k + w^{q-r+1} y^{q-l+1} z^{q^2-(l-1)^2} x^{F'(q)-F'(l-1)} P_{j-i}$
- 7: $P_k \leftarrow P_k + w^{q-(r-l)} y^q z^{q^2} x^{F'(q)-1} P_{j-i}$
- 8: **return** P_n

- j is the position of the rightmost symbol of the last long piece in $s_1[1 : k]$,
- i is the length of this long piece $s_1[j - i + 1 : j]$,
- l and r are the endpoints of a potential match of this long piece in s_2 ,
- q is the right endpoint of the block $s_2[r + 1 : q]$ that is cut into pieces of length one.

In short, we consider only long pieces in s_1 and every long piece $s_1[j - i + 1 : j]$ we correspond with a) block $s_2[l : q]$ that consists of $q - r + 1$ pieces: its match $s_2[l : r]$ and the group $s_2[r + 1 : q]$ of pieces of length one after it; b) block $s_2[1 : q]$ that consists of its match and two groups $s_2[1 : l - 1]$ and $s_2[r + 1 : q]$ of pieces of length one.

Now we need to show that s_1 and s_2 have a common partition of size t if and only if P_n contains a monomial $w^t x^{F'(n)-1} y^n z^{n^2}$. The if part is provided by the following lemma:

► **Lemma 6.** *If there is a common partition of s_1 and s_2 of size t , then there is a monomial $w^t x^{F'(n)-1} y^n z^{n^2}$ in P_n .*

Proof. Let $(l_1, r_1), \dots, (l_d, r_d)$ be a set of long pieces in the common partition. Without loss of generality we may assume that all (l_i, r_i) are sorted, that is $\forall i : r_i \leq l_{i+1}$. Each piece (l_i, r_i) for $i > 1$ may contribute a monomial

$$w^{q_i-r_i+1} x^{F'(q_i)-F'(l_i-1)} y^{q_i-(l_i-1)} z^{q_i^2-(l_i-1)^2},$$

where $q_i = l_{i+1} - 1$, $q_d = n$, and (l_1, r_1) may contribute a monomial $w^{q_1-r_1+l_1} x^{F'(q_1)-1} y^{q_1} z^{q_1^2}$, where $q_1 = l_2 - 1$. Here, for $i = 1$ we choose the monomial that takes into account pieces of size one before and after the long block (hence, $q_1 - r_1 + l_1$ in the exponent of w) and for $i > 1$ we choose the monomial that takes into account pieces of size one only after the long block (hence, $q_i - r_i + 1$ in the exponent of w). Thus, the sum of all the exponents of w is equal to the number of pieces in common partition, that is, t .

Multiplying all monomials together, we obtain the following monomial present in P_n :

$$\begin{aligned} & w^{q_1-r_1+l_1} x^{F'(q_1)-1} y^{q_1} z^{q_1^2} \prod_{i=2}^d w^{q_i-r_i+1} x^{F'(q_i)-F'(l_i-1)} y^{q_i-(l_i-1)} z^{q_i^2-(l_i-1)^2} = \\ & = w^t x^{F'(n)-1} y^n z^{n^2}. \end{aligned}$$

Now we are getting to the tricky part. We need to prove that if P_n contains a monomial $w^t x^{F'(n)-1} y^n z^{n^2}$ then s_1 and s_2 have a common partition with k pieces. ◀

For every set $S = \{(l_1, r_1), \dots, (l_d, r_d) \mid (l_i, r_i) \subset [n]\}$ of d intervals let $m(S)$ denote a monomial

$$\begin{aligned} m(S) &:= \prod_{i=1}^d x^{F'(r_i)-F'(l_i-1)} y^{r_i-l_i+1} z^{r_i^2-(l_i-1)^2} = \\ &= x^{\sum_i F'(r_i)-F'(l_i-1)} y^{\sum_i r_i-l_i+1} z^{\sum_i r_i^2-(l_i-1)^2}. \end{aligned}$$

The point of introducing such notation is that polynomial P_n is simply a sum of monomials of the form $w^\alpha m(S)$ for some $S = \{(l_1, r_1), \dots, (l_d, r_d)\}$ where all the intervals are long, so the $m(S)$ part relates to *what* is covered in s_2 and w^α part relates to *how* it is covered (that is, how many blocks).

It is easy to check that if there are no intersecting intervals in S and they cover the whole $[n]$ then $m(S) = x^{F'(n)-1} y^n z^{n^2}$ and we have a common partition of size α . What about intersecting intervals? The following lemma deals with this case:

► **Lemma 7.** *If S contains intersecting intervals and $m(S) = x^{F'(n)-1} y^n z^{n^2}$ then there is a set S' of non-intersecting intervals such that $m(S') = x^{F'(n)-1} y^\alpha z^\beta$ and $(\alpha < n) \vee (\beta < n^2)$.*

Proof. Suppose there are two long intersecting intervals (l_1, r_1) and (l_2, r_2) in S . We can show that there is a set of long non intersecting intervals $(l'_1, r'_1), \dots, (l'_p, r'_p)$ such that:

$$\sum_{j=1}^p (F'(r'_j) - F'(l'_j - 1)) = (F'(r_1) - F'(l_1 - 1)) + (F'(r_2) - F'(l_2 - 1)),$$

and one of the following two statements is true:

- $\sum_{j=1}^p (r'_j - l'_j + 1) < (r_1 - l_1 + 1) + (r_2 - l_2 + 1)$,
- $\sum_{j=1}^p (r'_j - l'_j + 1) = (r_1 - l_1 + 1) + (r_2 - l_2 + 1)$, but $\sum_{j=1}^p (r'^2_j - (l'_j - 1)^2) < (r_1^2 - (l_1 - 1)^2) + (r_2^2 - (l_2 - 1)^2)$.

If we replace (l_1, r_1) and (l_2, r_2) with this intervals we obtain a set S_1 such that $m(S_1) = y^\alpha z^\beta x^{F'(n)}$ and $(\alpha < n) \vee (\beta < n^2)$. We can keep replacing intersecting pairs with non-intersecting intervals preserving the value of \deg_x . It is clear that this process will eventually stop since $\deg_y \geq 0$ and $\deg_z \geq 0$, and thus the resulting set S' is well-defined and consists of non-intersecting intervals.

It remains to present the set $\{(l'_1, r'_1), \dots, (l'_p, r'_p)\}$ for every two intersecting intervals $\{(l_1, r_1), (l_2, r_2)\}$.

Let $T = \{(a_i, b_i), \dots, (a_p, b_p)\}$, such that $\forall i \in [p] : 0 \leq a_i < b_i - 1 \leq n - 1$ or $a_i = b_i$ (the later represents an interval of size 0 and is used only to reduce the number of cases in the analysis). We will use the following notation:

- $f(T) = \sum_{i=1}^{|T|} F'(b_i) - F'(a_i)$,
- $g(T) = \sum_{i=1}^{|T|} b_i - a_i$,
- $h(T) = \sum_{i=1}^{|T|} b_i^2 - a_i^2$.

Let A be a set $\{(a, b), (c, d)\}$ such that $(b \geq d > a \geq c) \vee (b \geq d > c \geq a)$. As there is no difference in analysis (we may replace $(a, b), (c, d)$ with $(a, d), (b, c)$), we consider only the first case. We are going to go through rigorous case analysis to show that we can always construct a set B such that $f(A) = f(B)$ and either $g(A) > g(B)$ or $g(A) = g(B)$ but $h(A) > h(B)$:

1. If $d > a + 2 \vee d = a + 1$ and $a > c + 2 \vee a = c + 1$ then

$$B = \{(c, a - 1), (a + 1, d), (b - 1, b + 1)\},$$

$$g(B) = d - c < b - a + d - c = g(A);$$

2. If $d > a + 2 \vee d = a + 1$ and $a = c + 2$ then

$$B = \{(c - 2, c), (a + 1, d), (b - 1, b + 1)\},$$

$$g(B) = d - a + 3 < b - a + d - c = g(A);$$

3. If $d = a + 2$ then

$$B = \{(c, d - 1), (b - 1, b + 1)\},$$

$$g(B) = d - a + 1 < b - a + d - c = g(A);$$

4. If $a = c$ and $d > a + 3 \vee d = a + 2$ and $b - a > 2$ then

$$B = \{(a - 2, a), (a + 2, d), (b - 1, b + 1)\},$$

$$g(B) = d - a + 2 < b - a + d - c = g(A);$$

5. If $a = c$ and $b - a = 2$ then

$$B = \{(a, b + 1)\},$$

$$g(B) = b - a + 1 < b - a + d - c = g(A);$$

6. If $a = c$ and $d = a + 3$ and $b > a + 3$ then

$$B = \{(a - 2, d - 1), (b - 1, b + 1)\},$$

$$g(B) = d - a + 3 < b - a + d - c = g(A);$$

7. If $a = c$ and $d = b$ and $b = a + 3$

$$B = \{(a - 2, b + 1)\},$$

$$g(B) = d - a + 3 = b - a + d - c = g(A),$$

$$h(B) = (a + 4)^2 - (a - 2)^2 = 12a + 12 < 12a + 18 = 2((a + 3)^2 - a^2) = h(A).$$

In some of the cases, it may turn out that for some interval from B its left endpoint is negative or its right endpoint is greater than n . In fact, the latter case is impossible: if long interval (a, b) covers at least $n + 1$ then $F'(b) - F'(a) > F'(n) - 1$. In the former case we will use the additional manipulations with B :

2. If $c = 0$ then we drop $(-2, 0)$. If $c = 1$ then we replace $(-1, 1)$ with $(0, 2)$;
 4. If $a = 0$ then we drop $(-2, 0)$. If $a = 1$ then we replace $(-1, 1)$ with $(0, 2)$;
 6. If $a = 0$ then we replace $(-2, 2)$ with $(0, 2)$. If $a = 1$ then we replace $(-1, 3)$ with $(2, 4)$;
 7. If $a = 0$ then we replace $(-2, 4)$ with $(0, 4)$. If $a = 1$ then we replace $(-1, 5)$ with $(4, 6)$.

Now to get the desired result we need to apply this case analysis to $\{(l_1 - 1, r_1), (l_2 - 1, r_2)\}$, get a set $B = \{(a'_1, b'_1) \dots (a'_p, b'_p)\}$ and get a collection $(a'_1 + 1, b'_1), \dots, (a'_p + 1, b'_p)$ which would satisfy all the intended properties. ◀

Suppose that there exists a set of intervals S with intersections and $m(S) = x^{F'(n)-1}y^n z^{n^2}$. Let S' be the corresponding set from the statement of the lemma. Since S' contains only non-intersecting intervals, $\deg_x(m(S')) \leq F'(n) - 1$ and equality can be achieved only if this intervals cover the whole $[n]$. If so, then $\deg_y(m(S')) = n = \deg_y(m(S))$ and $\deg_z(m(S')) = n^2 = \deg_z(m(S))$. But at least one of \deg_y and \deg_z decreases while we go from S to S' , which is a contradiction. Thus, the monomial $w^t x^{F'(n)-1} y^n z^{n^2}$ may only correspond to some set without intersections that covers the whole $[n]$ and in turn corresponds to a common partition of size t .

Finally, we can solve MCSP problem by finding the smallest t such that $w^t x^{F'(n)-1} y^n z^{n^2}$ is present in P_n . As we already have a polynomial time algorithm for P_n , its coefficients are not greater than $(2n^3)^n = 2^{O(n \log n)}$ and total degree is $O^*(\phi^n)$, by Theorem 2 there is $O^*(\phi^n)$ time and polynomial space algorithm for MCSP. ◀

5 Subexponential running time for small alphabets: hybrid strategy

For the case of a constant size alphabet, we provide a subexponential time algorithm, that collects information about partitions of s_1 and s_2 separately and then just checks if they have any partition in common. The main idea is to divide all the pieces in a partition into two groups: long ones and short ones. The good thing about long ones is that there are not too many of those due to their length. The good thing about short ones is that there are not too many possible short strings over our alphabet due to its small size. By balancing these two cases, we get the desired running time.

► **Theorem 8.** *MCSP can be solved in time and space $2^{O\left(\frac{n \log |\Sigma| \log \log n}{\log n}\right)}$ when $|\Sigma| = n^{o(1)}$.*

Proof. Let μ be a parameter whose value we will choose later. For every multiset S of strings no longer than μ , we define a *histogram* $\text{hist}_\mu(S) = \{(s, \text{count}(S, s)) \mid s \in \Sigma^*, |s| \leq \mu\}$, where $\text{count}(S, s)$ stands for a number of occurrences of s in S . As before, for both input strings we construct sets of their partitions, but now we treat each partition \mathcal{P} as a pair $(L, \text{hist}_\mu(S))$ of a multiset of long pieces $L = \{s \in \mathcal{P}, |s| > \mu\}$ and a histogram of a multiset of short ones $S = \{s \in \mathcal{P}, |s| \leq \mu\}$.

The construction of all possible partitions goes as follows: we iterate over all possible multisets of long pieces and for each of them we compute all possible histograms of short ones. As the number of long pieces is not greater than n/μ , there are at most $\binom{2n}{2n/\mu}$ ways to choose their ends.

Once we have fixed the long pieces, we want to enumerate all possible ways to cut the rest into small pieces. In order to do this, we use dynamic programming. Let s_1, s_2, \dots, s_k be the multiset of strings after removing all the long pieces. By $R(\{s_1, s_2, \dots, s_k\})$ we define the set of all possible ways to cut these strings into pieces no longer than μ . Then,

$$\begin{aligned} R(\{s_1, s_2, \dots, s_k\}) &= \\ &= \bigcup_{i=1}^{\min\{\mu, |s_k|\}} \{C \cup s_k[|s_k| - i + 1 : |s_k|] \mid C \in R(\{s_1, s_2, \dots, s_k[1 : |s_k| - i]\})\}, \end{aligned}$$

and $R(\{s_1, s_2, \dots, s_k\})$ may be computed in $O^*(n^{|\Sigma|^\mu})$, as $n^{|\Sigma|^\mu}$ is the upper bound on the number of all possible histograms: $\text{count}(S, s) \leq n$ for every $s \in S$ and there are at most $|\Sigma|^{\mu+1}$ distinct s , and it takes polynomial in n and linear in size time to obtain R if all smaller ones are already computed.

Thus, the total time is $O^*\left(\binom{2n}{2n/\mu} \cdot n^{|\Sigma|^\mu}\right)$ and we wish to choose μ so that this time is small as possible. Since $\binom{2n}{2n/\mu} = 2^{O\left(\frac{n}{\mu} \log \mu\right)}$ and $n^{|\Sigma|^\mu} = 2^{O(|\Sigma|^\mu \log n)}$, we may choose $\mu = c \log_2 n / \log_2 |\Sigma|$ with $c < 1$, and then the total time is $2^{O\left(\frac{n \log |\Sigma| \log \log n}{\log n}\right)}$. ◀

6 Lower bounds: reductions to MIS and to binary alphabet

In this section, we prove the ETH based lower bounds: assuming Exponential Time Hypothesis, the best upper bounds for MCSP and MCSP^c are $2^{\Omega(n)}$ and $2^{\Omega(n/\log n)}$, respectively.

► **Lemma 9.** *Assuming ETH, there is no $2^{o(n)}$ time algorithm for MCSP.*

Proof. [8] proves NP-hardness of MCSP by reducing the maximum independent set on degree-3 graphs (3-MIS) to MCSP. The reduction is linear: given a graph with n nodes, it produces an instance of MCSP with strings of length $O(n)$. In turn, [12] shows that there is no $2^{o(n)}$ time algorithm for 3-MIS under ETH. This implies that the same is true for MCSP. ◀

► **Theorem 10.** *Assuming ETH, there is no $2^{o(n/\log n)}$ time algorithm for MCSP on constant size alphabets.*

Proof. We show how to reduce an instance of the general MCSP of length n to an instance of MCSP over binary alphabet that is $4 \log n$ times longer. Since the general MCSP cannot be solved in time $2^{o(n)}$ under ETH, we obtain a lower bound $2^{\Omega(n/\log n)}$ for its constant-size alphabet version.

Consider an instance of MCSP that consists of strings s_1 and s_2 of length n . Since there are at most n distinct symbols in both strings, we can encode each symbol c via binary string $b(c)$ of length $\log_2 n$. For every symbol c we define a *gadget* $\psi(c)$ as a string

$$\psi(c) = 0 \, b(c)[1] \, 0 \, b(c)[2] \, 0 \dots 0 \, b(c)[\log_2 n] \, 0 \, 1 \, 1 \, 0 \, b(c)[1] \, 0 \, b(c)[2] \, 0 \dots 0 \, b(c)[\log_2 n] \, 0.$$

We call the central 1 of a gadget its *pivot*.

We transform $s_1 = s_1[1]s_1[2] \dots s_1[n]$ into a string $s'_1 = \psi(s_1[1])\psi(s_1[2]) \dots \psi(s_1[n])$ and $s_2 = s_2[1]s_2[2] \dots s_2[n]$ into a string $s'_2 = \psi(s_2[1])\psi(s_2[2]) \dots \psi(s_2[n])$. It is clear that (s'_1, s'_2) is an instance of MCSP over binary alphabet of size $n \cdot \lceil 4 \log_2 n + 5 \rceil$. We show that there is a common partition of s_1 and s_2 of size at most t if and only if there is a common partition of s'_1 and s'_2 of size at most t .

Necessity. Given a common partition of s_1 and s_2 , we can transform it into a common partition of s'_1 and s'_2 of the same size simply by replacing each block $b = b[1]b[2] \dots b[b]$ with $\psi(b) = \psi(b[1])\psi(b[2]) \dots \psi(b[b])$.

Sufficiency. Consider a common partition of s'_1 and s'_2 of size t . Call the block of this partition *long* if it contains at least two pivots of some gadgets. Since the pivot and its neighbors are the only consecutive 1's in gadget, if some long block b_1 of s'_1 matched with some long block b_2 of s'_2 then all the pivots of b_1 matched with the corresponding pivots of b_2 . Note that long blocks, along with each pivot, contain information about the corresponding symbols of its gadgets, as this information is duplicated on both sides of the pivot. This means that we can correspond every long block that contains pivots of gadgets $\psi(c[1])\psi(c[2]) \dots \psi(c[c])$ with substring $c[1]c[2] \dots c[c]$ of s_1 and s_2 .

Now we are ready to present a common partition of s_1 and s_2 of size at most t : for every long block in (s'_1, s'_2) we take the corresponding block in (s_1, s_2) and cut the remains into blocks of size one. Clearly, the result is a common partition of s_1 and s_2 : blocks of length at least two do not intersect and are contained in both strings since so do the corresponding long blocks. The remaining blocks of size one match since s_1 and s_2 consists of the same number of each symbol. The size of the constructed partition is equal to the number of blocks in the considered partition of s'_1 and s'_2 that contain at least one pivot and hence is no more than t . ◀

7 Open problems

There are three natural questions left open by the present studies.

1. Close the gap between a lower bound $2^{\Omega(n/\log n)}$ and an upper bound $2^{O(n \log \log n / \log n)}$ for constant size alphabets.
2. Design a moderately exponential time (of the form c^n for $c < 2$) algorithm for the case of more than two input strings.
3. Roughly, it is the “no-holes” property that allows us to improve the 2^n upper bound for MCSP. What are other problems with the same effect?

References

- 1 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.
- 2 Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In Aaron E. Darling and Jens Stoye, editors, *Algorithms in Bioinformatics - 13th International Workshop, WABI 2013, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8126 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2013. doi:10.1007/978-3-642-40453-5_19.
- 3 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 102–121. SIAM, 2014. doi:10.1137/1.9781611973402.8.
- 4 Marek Chrobak, Petr Kolman, and Jiri Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Trans. Algorithms*, 1(2):350–366, 2005. doi:10.1145/1103963.1103971.
- 5 Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms*, 3(1):2:1–2:19, 2007. doi:10.1145/1219944.1219947.
- 6 Peter Damaschke. Minimum common string partition parameterized. In Keith A. Crandall and Jens Lagergren, editors, *Algorithms in Bioinformatics, 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings*, volume 5251 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2008. doi:10.1007/978-3-540-87361-7_8.
- 7 Bin Fu, Haitao Jiang, Boting Yang, and Binhai Zhu. Exponential and polynomial time algorithms for the minimum common string partition problem. In Weifan Wang, Xuding Zhu, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications - 5th International Conference, COCOA 2011, Zhangjiajie, China, August 4-6, 2011. Proceedings*, volume 6831 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2011. doi:10.1007/978-3-642-22616-8_24.
- 8 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. *Electron. J. Comb.*, 12, 2005. URL: http://www.combinatorics.org/Volume_12/Abstracts/v12i1r50.html.
- 9 Isaac Goldstein and Moshe Lewenstein. Quick greedy computation for minimum common string partition. *Theor. Comput. Sci.*, 542:98–107, 2014. doi:10.1016/j.tcs.2014.05.006.
- 10 Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Families with infants: Speeding up algorithms for NP-hard problems using FFT. *ACM Trans. Algorithms*, 12(3):35:1–35:17, 2016. doi:10.1145/2847419.
- 11 Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *J. Comb. Optim.*, 23(4):519–527, 2012. doi:10.1007/s10878-010-9370-2.
- 12 David S. Johnson and Mario Szegedy. What are the least tractable instances of max independent set? In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 927–928. ACM/SIAM, 1999. URL: <http://dl.acm.org/citation.cfm?id=314500.315093>.
- 13 Haim Kaplan and Nira Shafir. The greedy algorithm for edit distance with moves. *Inf. Process. Lett.*, 97(1):23–27, 2006. doi:10.1016/j.ipl.2005.08.010.
- 14 Mikko Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Inf. Process. Lett.*, 98(1):24–28, 2006. doi:10.1016/j.ipl.2005.11.013.
- 15 Petr Kolman and Tomasz Walen. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electron. J. Comb.*, 14(1), 2007. URL: http://www.combinatorics.org/Volume_14/Abstracts/v14i1r50.html.
- 16 Leopold Kronecker. Grundzüge einer arithmetischen theorie der algebraischen grössen. *J. Reine Angew. Math.*, 92:1–122, 1882.

- 17 Dana Shapira and James A. Storer. Edit distance with move operations. *J. Discrete Algorithms*, 5(2):380–392, 2007. doi:10.1016/j.jda.2005.01.010.
- 18 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

An Accelerated Newton–Dinkelbach Method and Its Application to Two Variables per Inequality Systems

Daniel Dadush 

CWI, Amsterdam, The Netherlands

Zhuan Khye Koh 

Department of Mathematics, London School of Economics and Political Science, UK

Bento Natura 

Department of Mathematics, London School of Economics and Political Science, UK

László A. Végh 

Department of Mathematics, London School of Economics and Political Science, UK

Abstract

We present an accelerated, or “look-ahead” version of the Newton–Dinkelbach method, a well-known technique for solving fractional and parametric optimization problems. This acceleration halves the Bregman divergence between the current iterate and the optimal solution within every two iterations. Using the Bregman divergence as a potential in conjunction with combinatorial arguments, we obtain strongly polynomial algorithms in three applications domains: (i) For linear fractional combinatorial optimization, we show a convergence bound of $O(m \log m)$ iterations; the previous best bound was $O(m^2 \log m)$ by Wang et al. (2006). (ii) We obtain a strongly polynomial label-correcting algorithm for solving linear feasibility systems with two variables per inequality (2VPI). For a 2VPI system with n variables and m constraints, our algorithm runs in $O(mn)$ iterations. Every iteration takes $O(mn)$ time for general 2VPI systems, and $O(m + n \log n)$ time for the special case of deterministic Markov Decision Processes (DMDPs). This extends and strengthens a previous result by Madani (2002) that showed a weakly polynomial bound for a variant of the Newton–Dinkelbach method for solving DMDPs. (iii) We give a simplified variant of the parametric submodular function minimization result by Goemans et al. (2017).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Mathematical optimization

Keywords and phrases Newton–Dinkelbach method, fractional optimization, parametric optimization, strongly polynomial algorithms, two variables per inequality systems, Markov decision processes, submodular function minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.36

Related Version *Full Version:* <https://arxiv.org/abs/2004.08634>

Funding This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement nos. 757481–ScaleOpt and 805241–QIP).

Acknowledgements The fourth author would like to thank Neil Olver for several inspiring discussions on 2VPI systems, in particular, on symmetries of the problem.

1 Introduction

Linear fractional optimization problems are well-studied in combinatorial optimization. Given a closed domain $\mathcal{D} \subseteq \mathbb{R}^m$ and $c, d \in \mathbb{R}^m$ such that $d^\top x > 0$ for all $x \in \mathcal{D}$, the problem is

$$\inf c^\top x / d^\top x \quad \text{s.t. } x \in \mathcal{D}. \quad (1)$$



© Daniel Dadush, Zhuan Khye Koh, Bento Natura, and László A. Végh;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 36; pp. 36:1–36:15



Leibniz International Proceedings in Informatics

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

The domain \mathcal{D} could be either a convex set or a discrete set $\mathcal{D} \subseteq \{0, 1\}^m$. Classical examples include finding minimum cost-to-time ratio cycles and minimum ratio spanning trees. One can equivalently formulate (1) as a parametric search problem. Let

$$f(\delta) = \inf\{(c - \delta d)^\top x : x \in \mathcal{D}\}, \quad (2)$$

be a concave and decreasing function. Assuming (1) has a finite optimum δ , it corresponds to the unique root $f(\delta) = 0$.

A natural question is to investigate how the computational complexity of solving the minimum ratio problem (1) may depend on the complexity of the corresponding linear optimization problem $\min c^\top x$ s.t. $x \in \mathcal{D}$. Using the reformulation (2), one can reduce the fractional problem to the linear problem via binary search; however, the number of iterations needed to find an exact solution may depend on the bit complexity of the input. A particularly interesting question is: assuming there exists a strongly polynomial algorithm for linear optimization over a domain \mathcal{D} , can we find a strongly polynomial algorithm for linear fractional optimization over the same domain?

A seminal paper by Megiddo [14] introduced the *parametric search* technique to solve linear fractional combinatorial optimization problems. He showed that if the linear optimization algorithm only uses $p(m)$ comparisons and $q(m)$ additions, then there exists an $O(p(m)(p(m) + q(m)))$ algorithm for the linear fractional optimization problem. This in particular yielded the first strongly polynomial algorithm for the minimum cost-to-time ratio cycle problem. On a very high level, parametric search works by simulating the linear optimization algorithm for the parametric problem (2), with the parameter $\delta \in \mathbb{R}$ being indeterminate.

A natural alternative approach is to solve (2) using a standard root finding algorithm. Radzik [18] showed that for a discrete domain $\mathcal{D} \subseteq \{0, 1\}^m$, the *discrete Newton* method – in this context, also known as *Dinkelbach’s method* [4] – terminates in a strongly polynomial number of iterations. In contrast to parametric search, there are no restrictions on the possible operations in the linear optimization algorithm. In certain settings, such as the maximum ratio cut problem, the discrete Newton method outperforms parametric search; we refer to the comprehensive survey by Radzik [19] for details and comparison of the two methods.

1.1 Our Contributions

We introduce a new, *accelerated variant of Newton’s method for univariate functions*. Let $f : \mathbb{R} \rightarrow \mathbb{R} \cup \{-\infty\}$ be a concave function. Under some mild assumptions on f , our goal is to either find the largest root, or show that no root exists. Let δ^* denote the largest root, or in case $f < 0$, let δ^* denote the largest maximizer of f . For simplicity, we now describe the method for differentiable functions. This will not hold in general: functions of the form (2) will be piecewise linear if \mathcal{D} is finite or polyhedral. The algorithm description in Section 2 uses a form with supergradients (that can be chosen arbitrarily between the left and right derivatives).

The standard Newton method, also used by Radzik, proceeds through iterates $\delta^{(1)} > \delta^{(2)} > \dots > \delta^{(t)}$ such that $f(\delta^{(i)}) \leq 0$, and updates $\delta^{(i+1)} = \delta^{(i)} - f(\delta^{(i)})/f'(\delta^{(i)})$.

Our new variant uses a more aggressive “*look-ahead*” technique. At each iteration, we compute $\delta = \delta^{(i)} - f(\delta^{(i)})/f'(\delta^{(i)})$, and jump ahead to $\delta' = 2\delta - \delta^{(i)}$. In case $f(\delta') \leq 0$ and $f'(\delta') < 0$, we update $\delta^{(i+1)} = \delta'$; otherwise, we continue with the standard iterate δ .

This modification leads to an improved and at the same time simplified analysis based on the *Bregman divergence* $D_f(\delta^*, \delta^{(i)}) = f(\delta^{(i)}) + f'(\delta^{(i)})(\delta^* - \delta^{(i)}) - f(\delta^*)$. We show that this decreases by a factor of two between any two iterations.

A salient feature of the algorithm is that it handles both feasible and infeasible outcomes in a unified framework. In the context of linear fractional optimization, this means that the assumption $d^\top x > 0$ for all $x \in \mathcal{D}$ in (1) can be waived. Instead, $d^\top x > 0$ is now added as a feasibility constraint to (1). This generalization is important when we use the algorithm to solve two variables per inequality systems.

This general result leads to improvements and simplifications of a number of algorithms using the discrete Newton method.

- For *linear fractional combinatorial optimization*, namely the setting (1) with $\mathcal{D} \subseteq \{0, 1\}^m$, we obtain an $O(m \log m)$ bound on the number of iterations, a factor m improvement over the previous best bound $O(m^2 \log m)$ by Wang et al. [25] from 2006. We remark that Radzik's first analysis [18] yielded a bound of $O(m^4 \log^2 m)$ iterations, improved to $O(m^2 \log^2 m)$ in [19].
- Goemans et al. [7] used the discrete Newton method to obtain a strongly polynomial algorithm for parametric submodular function minimization. We give a simple new variant of this result with the same asymptotic running time, using the accelerated algorithm.
- For *two variable per inequality (2VPI)* systems, we obtain a *strongly polynomial label-correcting algorithm*. This will be discussed in more detail next.

1.2 Two Variables Per Inequality Systems

A major open question in the theory of linear programming (LP) is whether there exists a strongly polynomial algorithm for LP. This problem is one of Smale's eighteen mathematical challenges for the twenty-first century [22]. An LP algorithm is *strongly polynomial* if it only uses elementary arithmetic operations ($+$, $-$, \times , $/$) and comparisons, and the number of such operations is polynomially bounded in the number of variables and constraints. Furthermore, the algorithm needs to be in PSPACE, i.e. the numbers occurring in the computations must remain polynomially bounded in the input size.

The notion of a strongly polynomial algorithm was formally introduced by Megiddo [15] in 1983 (using the term “*genuinely polynomial*”), where he gave the first such algorithm for *two variables per inequality (2VPI)* systems. These are feasibility LPs where every inequality contains at most two variables. More formally, let $\mathcal{M}_2(n, m)$ be the set of $n \times m$ matrices with at most two nonzero entries per column. A 2VPI system is of the form $A^\top y \leq c$ for $A \in \mathcal{M}_2(n, m)$ and $c \in \mathbb{R}^m$.

If we further require that every inequality has at most one positive and at most one negative coefficient, it is called a *monotone two variables per inequality (M2VPI)* system. A simple and efficient reduction is known from 2VPI systems with n variables and m inequalities to M2VPI systems with $2n$ variables and $\leq 2m$ inequalities [5, 10].

Connection between 2VPI and parametric optimization. An M2VPI system has a natural graphical interpretation: after normalization, we can assume every constraint is of the form $y_u - \gamma_e y_v \leq c_e$. Such a constraint naturally maps to an arc $e = (u, v)$ with *gain factor* $\gamma_e > 0$ and cost c_e . Based on Shostak's work [21] that characterized feasibility in terms of this graph, Aspvall and Shiloach [2] gave the first weakly polynomial algorithm for M2VPI systems.

We say that a directed cycle C is *flow absorbing* if $\prod_{e \in C} \gamma_e < 1$ and *flow generating* if $\prod_{e \in C} \gamma_e > 1$. Every flow absorbing cycle C implies an upper bound for every variable y_u incident to C ; similarly, flow generating cycles imply lower bounds. The crux of Aspvall and Shiloach's algorithm is to find the tightest upper and lower bounds for each variable y_u .

Finding these bounds corresponds to solving fractional optimization problems of the form (1), where $\mathcal{D} \subseteq \mathbb{R}^m$ describes “generalized flows” around cycles. The paper [2] introduced the *Grapevine* algorithm – a natural modification the Bellman-Ford algorithm – to decide whether the optimum ratio is smaller or larger than a fixed value δ . The optimum value can be found using binary search on the parameter.

Megiddo’s strongly polynomial algorithm [15] replaced the binary search framework in Aspvall and Shiloach’s algorithm by extending the parametric search technique in [14]. Subsequently, Cohen and Megiddo [3] devised faster strongly polynomial algorithms for the problem. The current fastest strongly polynomial algorithm is given by Hochbaum and Naor [11], an efficient Fourier–Motzkin elimination with running time of $O(mn^2 \log m)$.

2VPI via Newton’s method. Since Newton’s method proved to be an efficient and viable alternative to parametric search, a natural question is to see whether it can solve the parametric problems occurring in 2VPI systems. Radzik’s fractional combinatorial optimization results [18, 19] are not directly applicable, since the domain \mathcal{D} in this setting is a polyhedron and not a discrete set.¹ Madani [13] used a variant of the Newton–Dinkelbach method as a tool to analyze the convergence of policy iteration on *deterministic Markov Decision Processes (DMDPs)*, a special class of M2VPI systems (discussed later in more detail). He obtained a weakly polynomial convergence bound; it remained open whether such an algorithm could be strongly polynomial.

Our 2VPI algorithm. We introduce a new type of strongly polynomial 2VPI algorithm by combining the accelerated Newton–Dinkelbach method with a “*variable fixing*” analysis. Variable fixing was first introduced in the seminal work of Tardos [23] on minimum-cost flows, and has been a central idea of strongly polynomial algorithms, see in particular [8, 20] for cycle cancelling minimum-cost flow algorithms, and [16, 24] for maximum generalized flows, a dual to the 2VPI problem.

We show that for every iterate $\delta^{(i)}$, there is a constraint that has been “actively used” at $\delta^{(i)}$ but will not be used ever again after a strongly polynomial number of iterations. The analysis combines the decay in *Bregman divergence* shown in the general accelerated Newton–Dinkelbach analysis with a combinatorial “*subpath monotonicity*” property.

Our overall algorithm can be seen as an extension of Madani’s DMDP algorithm. In particular, we adapt his “unfreezing” idea: the variables y_u are admitted to the system one-by-one, and the accelerated Newton–Dinkelbach method is used to find the best “cycle bound” attainable at the newly admitted y_u in the graph induced by the current variable set. This returns a feasible solution or reports infeasibility within $O(m)$ iterations. As every iteration takes $O(mn)$ time, our overall algorithm terminates in $O(m^2n^2)$ time. For the special setting of deterministic MDPs, the runtime per iteration improves to $O(m + n \log n)$, giving a total runtime of $O(mn(m + n \log n))$.

Even though our running time bound is worse than the state-of-the-art 2VPI algorithm [11], it is of a very different nature from all previous 2VPI algorithms. In fact, our algorithm is a *label-correcting algorithm*, naturally fitting to the family of algorithms used in other combinatorial optimization problems with constraint matrices from $\mathcal{M}_2(n, m)$ such as maximum flow, shortest paths, minimum-cost flow, and generalized flow problems. We next elaborate on this connection.

¹ The problem could be alternatively formulated with $\mathcal{D} \subseteq \{0, 1\}^m$ but with nonlinear functions instead of $c^\top x$ and $d^\top x$.

Label-correcting algorithms. An important special case of M2VPI systems corresponds to the shortest paths problem: given a directed graph $G = (V, E)$ with target node $t \in V$ and arc costs $c \in \mathbb{R}^E$, we associate constraints $y_u - y_v \leq c_e$ for every arc $e = (u, v) \in E$ and $y_t = 0$. If the system is feasible and bounded, the pointwise maximal solution corresponds to the shortest path labels to t ; an infeasible system contains a negative cost cycle. A generic label-correcting algorithm maintains distance labels y that are upper bounds on the shortest path distances to t . The labels are decreased according to violated constraints. Namely, if $y_u - y_v > c_e$, then decreasing y_u to $c_e + y_v$ gives a smaller valid distance label at u . We terminate with the shortest path labels once all constraints are satisfied. The Bellman–Ford algorithm for the shortest paths problem is a particular implementation of the generic label-correcting algorithm; we refer the reader to [1, Chapter 5] for more details.

It is a natural question if label-correcting algorithms can be extended to general M2VPI systems, where constraints are of the form $y_u - \gamma_e y_v \leq c_e$ for a “gain/loss factor” $\gamma_e > 0$ associated with each arc. A fundamental property of M2VPI systems is that, whenever bounded, a unique pointwise maximal solution exists, i.e. a feasible solution y^* such that $y \leq y^*$ for every feasible solution y . A label-correcting algorithm for such a setting can be naturally defined as follows. Let us assume that the problem is bounded. The algorithm should proceed via a decreasing sequence $y^{(0)} \geq y^{(1)} \geq \dots \geq y^{(t)}$ of labels that are all valid upper bounds on any feasible solution y to the system. The algorithm either terminates with the unique pointwise maximal solution $y^{(t)} = y^*$, or finds an infeasibility certificate.

The basic label-correcting operation is the “arc update”, decreasing y_u to $\min\{y_u, c_e + \gamma_e y_v\}$ for some arc $e = (u, v) \in E$. Such updates suffice in the shortest path setting. However, in the general setting arc operations only may not lead to finite termination. Consider a system with only two variables, y_u and y_v , and two constraints, $y_u - y_v \leq 0$, and $y_v - \frac{1}{2}y_u \leq -1$. The alternating sequence of arc updates converges to $(y_u^*, y_v^*) = (-2, -2)$, but does not finitely terminate. In this example, we can “detect” the cycle formed by the two arcs, that implies the bound $y_u - \frac{1}{2}y_u \leq -1$.

Shostak’s [21] result demonstrates that arc updates, together with such “cycle updates” should be sufficient for finite termination. Our M2VPI algorithm amounts to the first strongly polynomial label-correcting algorithm for general M2VPI systems, using arc updates and cycle updates.

Deterministic Markov decision processes. A well-studied special case of M2VPI systems in which $\gamma \leq 1$ is known as *deterministic Markov decision process* (DMDP). A *policy* corresponds to selecting an outgoing arc from every node, and the objective is to find a policy that minimizes the total discounted cost over an infinite time horizon. The pointwise maximal solution of this system corresponds to the optimal values of a policy.

The standard policy iteration, value iteration, and simplex algorithms can be all interpreted as variants of the label-correcting framework.²

Value iteration can be seen as a generalization of the Bellman–Ford algorithm to the DMDP setting. As our previous example shows, value iteration may not be finite. One could still consider as the termination criterion the point where value iteration “reveals” the optimal policy, i.e. updates are only performed using constraints that are tight in the optimal solution. If each discount factor γ_e is at most γ' for some $\gamma' > 0$, then it is well-known that value iteration converges at the rate $1/(1 - \gamma')$. This is in fact true more generally, for nondeterministic MDPs [12]. However, if the discount factors can be arbitrarily close to 1,

² The value sequence may violate monotonicity in certain cases of value iteration.

then Feinberg and Huang [6] showed that value iteration cannot reveal the optimal policy in strongly polynomial time even for DMDPs. Post and Ye [17] proved that simplex with the most negative reduced cost pivoting rule is strongly polynomial for DMDPs; this was later improved by Hansen et al. [9]. These papers heavily rely on the assumption $\gamma \leq 1$, and does not seem to extend to general M2VPI systems.

Madani’s previously mentioned work [13] used a variant of the Newton–Dinkelbach method as a tool to analyze the convergence of policy iteration on deterministic MDPs, and derived a weakly polynomial runtime bound.

Paper organization We start by giving preliminaries and introducing notation below. In Section 2, we present an accelerated Newton’s method for univariate concave functions, and apply it to linear fractional combinatorial optimization and linear fractional programming. Section 3 contains our main application of the method to the 2VPI problem. Our results on parametric submodular function minimization and deterministic MDPs can be found in the full version of the paper. Missing proofs also appear in the full version.

Preliminaries Let \mathbb{R}_+ and \mathbb{R}_{++} be the nonnegative and positive reals respectively, and denote $\bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$. Given a proper concave function $f : \mathbb{R} \rightarrow \bar{\mathbb{R}}$, let $\text{dom}(f) := \{x : -\infty < f(x) < \infty\}$ be the effective domain of f . For a point $x_0 \in \text{dom}(f)$, denote the set of supergradients of f at x_0 as $\partial f(x_0) := \{g : f(x) \leq f(x_0) + g(x - x_0) \forall x \in \mathbb{R}\}$. If x_0 is in the interior of $\text{dom}(f)$, then $\partial f(x_0) = [f'_-(x_0), f'_+(x_0)]$, where $f'_-(x_0)$ and $f'_+(x_0)$ are the left and right derivatives. Throughout, we use $\log(x) = \log_2(x)$ to indicate base 2 logarithm. For $x, y \in \mathbb{R}^m$, denote $x \circ y \in \mathbb{R}^m$ as the element-wise product of the two vectors.

2 An Accelerated Newton–Dinkelbach Method

Let $f : \mathbb{R} \rightarrow \bar{\mathbb{R}}$ be a proper concave function such that $f(\delta) \leq 0$ and $\partial f(\delta) \cap \mathbb{R}_{<0} \neq \emptyset$ for some $\delta \in \text{dom}(f)$. Given a suitable starting point, as well as value and supergradient oracles of f , the Newton–Dinkelbach method either computes the largest root of f or declares that it does not have a root. In this paper, we make the mild assumption that f has a root or attains its maximum. Consequently, the point $\delta^* := \max(\{\delta : f(\delta) = 0\} \cup \arg \max f(\delta))$ is well-defined. It is the largest root of f if f has a root. Otherwise, it is the largest maximizer of f . Therefore, the Newton–Dinkelbach method returns δ^* if f has a root, and certifies that $f(\delta^*) < 0$ otherwise.

The algorithm takes as input an initial point $\delta^{(1)} \in \text{dom}(f)$ and a supergradient $g^{(1)} \in \partial f(\delta^{(1)})$ such that $f(\delta^{(1)}) \leq 0$ and $g^{(1)} < 0$. At the start of every iteration $i \geq 1$, it maintains a point $\delta^{(i)} \in \text{dom}(f)$ and a supergradient $g^{(i)} \in \partial f(\delta^{(i)})$ where $f(\delta^{(i)}) \leq 0$. If $f(\delta^{(i)}) = 0$, then it returns $\delta^{(i)}$ as the largest root of f . Otherwise, a new point $\delta := \delta^{(i)} - f(\delta^{(i)})/g^{(i)}$ is generated. Now, there are two scenarios in which the algorithm terminates and reports that f does not have a root: (1) $f(\delta) = -\infty$; (2) $f(\delta) < 0$ and $g \geq 0$ where $g \in \partial f(\delta)$ is the supergradient given by the oracle. If both scenarios do not apply, the next point and supergradient is set to $\delta^{(i+1)} := \delta$ and $g^{(i+1)} := g$ respectively. Then, a new iteration begins.

To accelerate this classical method, we perform an aggressive guess $\delta' := 2\delta - \delta^{(i)} < \delta$ on the next point at the end of every iteration i . We call this procedure *look-ahead*, which is implemented on Lines 7–10 of Algorithm 1. Let $g' \in \partial f(\delta')$ be the supergradient returned by the oracle. If $-\infty < f(\delta') < 0$ and $g' < 0$, then the next point and supergradient are set to $\delta^{(i+1)} := \delta'$ and $g^{(i+1)} := g'$ respectively as $\delta' \geq \delta^*$. In this case, we say that look-ahead is *successful* in iteration i . Otherwise, we proceed as usual by taking $\delta^{(i+1)} := \delta$ and $g^{(i+1)} := g$.

Algorithm 1 LOOK-AHEADNEWTON.

input : Value and supergradient oracles for a proper concave function f , an initial point $\delta^{(1)} \in \text{dom}(f)$ and supergradient $g^{(1)} \in \partial f(\delta^{(1)})$ where $f(\delta^{(1)}) \leq 0$ and $g^{(1)} < 0$.

output : The largest root of f if it exists; report NO ROOT otherwise.

```

1  $i \leftarrow 1$ 
2 while  $f(\delta^{(i)}) < 0$  do
3    $\delta \leftarrow \delta^{(i)} - f(\delta^{(i)})/g^{(i)}$ 
4    $g \in \partial f(\delta)$  /* Empty if  $f(\delta) = -\infty$  */
5   if  $f(\delta) = -\infty$  or ( $f(\delta) < 0$  and  $g \geq 0$ ) then
6     return NO ROOT
7    $\delta' \leftarrow 2\delta - \delta^{(i)}$  /* Look-ahead guess */
8    $g' \in \partial f(\delta')$  /* Empty if  $f(\delta') = -\infty$  */
9   if  $-\infty < f(\delta') < 0$  and  $g' < 0$  then /* Is the guess successful? */
10     $\delta \leftarrow \delta', g \leftarrow g'$ 
11     $\delta^{(i+1)} \leftarrow \delta, g^{(i+1)} \leftarrow g$ 
12     $i \leftarrow i + 1$ 
13 return  $\delta^{(i)}$ 

```

It is easy to see that $\delta^{(i)}$ is monotonically decreasing while $f(\delta^{(i)})$ is monotonically increasing. Furthermore, $g^{(i)}$ is monotonically increasing except in the final iteration where it may remain unchanged (Lemma 2.1). Similarly, we have $g^{(i)} < 0$ except possibly in the final iteration when $f(\delta^{(i)}) = 0$.

► **Lemma 2.1.** *For every iteration $i \geq 2$, we have $\delta^* \leq \delta^{(i)} < \delta^{(i-1)}$, $f(\delta^*) \geq f(\delta^{(i)}) > f(\delta^{(i-1)})$ and $g^{(i)} \geq g^{(i-1)}$, where the last inequality holds at equality if and only if $g^{(i)} = \inf_{g \in \partial f(\delta^{(i)})} g$, $g^{(i-1)} = \sup_{g \in \partial f(\delta^{(i-1)})} g$ and $f(\delta^{(i)}) = 0$. Moreover,*

$$\frac{f(\delta^{(i)})}{f(\delta^{(i-1)})} + \frac{g^{(i)}}{g^{(i-1)}} \leq 1.$$

Our analysis of the Newton–Dinkelbach method utilizes the Bregman divergence associated with f as a potential. Even though the original definition requires f to be differentiable and strictly concave, it can be naturally extended to our setting in the following way.

► **Definition 2.2.** *Given a proper concave function $f : \mathbb{R} \rightarrow \bar{\mathbb{R}}$, the Bregman divergence associated with f is defined as*

$$D_f(\delta', \delta) := \begin{cases} f(\delta) + \sup_{g \in \partial f(\delta)} g(\delta' - \delta) - f(\delta') & \text{if } \delta \neq \delta', \\ 0 & \text{otherwise.} \end{cases}$$

for all $\delta, \delta' \in \text{dom}(f)$ such that $\partial f(\delta) \neq \emptyset$.

Since f is concave, the Bregman divergence is nonnegative. It is again easy to see that $D_f(\delta^*, \delta^{(i)})$ is monotonically decreasing except in the final iteration where it may remain unchanged (Lemma 2.3).

► **Lemma 2.3.** *For every iteration $i \geq 2$, we have $D_f(\delta^*, \delta^{(i)}) \leq D_f(\delta^*, \delta^{(i-1)})$ which holds at equality if and only if $g^{(i-1)} = \inf_{g \in \partial f(\delta^{(i-1)})} g$ and $f(\delta^{(i)}) = 0$.*

If look-ahead is successful, then we have made significant progress. Otherwise, by our choice of δ' , we learn that we are not too far away from δ^* . The next lemma demonstrates the advantage of using the look-ahead Newton–Dinkelbach method. It exploits the proximity to δ^* to produce a geometric decay in the Bregman divergence of $\delta^{(i)}$ and δ^* .

► **Lemma 2.4.** *For every iteration $i > 2$ in Algorithm 1, we have $D_f(\delta^*, \delta^{(i)}) < \frac{1}{2} D_f(\delta^*, \delta^{(i-2)})$.*

In the remaining of this section, we apply the accelerated Newton–Dinkelbach method to linear fractional combinatorial optimization and linear fractional programming. The application to parametric submodular function minimization is in the full version.

2.1 Linear Fractional Combinatorial Optimization

The problem (1) with $\mathcal{D} \subseteq \{0, 1\}^m$ is known as *linear fractional combinatorial optimization*. Radzik [18] showed that the Newton–Dinkelbach method applied to the function $f(\delta)$ as in (2) terminates in a strongly polynomial number of iterations. Recall that $f(\delta) = \min_{x \in \mathcal{D}} (c - \delta d)^\top x$. By the assumption $d^\top x > 0$ for all $x \in \mathcal{D}$, this function is concave, strictly decreasing, finite and piecewise-linear. Hence, it has a unique root. Moreover, $f(\delta) < 0$ and $\partial f(\delta) \cap \mathbb{R}_{<0} \neq \emptyset$ for sufficiently large δ . To implement the value and supergradient oracles, we assume that a linear optimization oracle over \mathcal{D} is available, i.e. it returns an element in $\arg \min_{x \in \mathcal{D}} (c - \delta d)^\top x$ for any $\delta \in \mathbb{R}$.

Our result for the accelerated variant improves the state-of-the-art bound $O(m^2 \log m)$ by Wang et al. [25] on the standard Newton–Dinkelbach method. We will need the following lemma, given by Radzik and credited to Goemans in [19]. It gives a strongly polynomial bound on the length of a geometrically decreasing sequence of sums.

► **Lemma 2.5 ([19]).** *Let $c \in \mathbb{R}_+^m$ and $x^{(1)}, x^{(2)}, \dots, x^{(k)} \in \{-1, 0, 1\}^m$. If $0 < c^\top x^{(i+1)} \leq \frac{1}{2} c^\top x^{(i)}$ for all $i < k$, then $k = O(m \log m)$.*

► **Theorem 2.6.** *Algorithm 1 converges in $O(m \log m)$ iterations for linear fractional combinatorial optimization problems.*

Proof. Observe that Algorithm 1 terminates in a finite number of iterations because f is piecewise linear. Let $\delta^{(1)} > \delta^{(2)} > \dots > \delta^{(k)} = \delta^*$ denote the sequence of iterates at the start of Algorithm 1. Since f is concave, we have $D_f(\delta^*, \delta^{(i)}) \geq 0$ for all $i \in [k]$. For each $i \in [k]$, pick $x^{(i)} \in \arg \min_{x \in \mathcal{D}} (c - \delta^{(i)} d)^\top x$ which maximizes $d^\top x$. This is well-defined because f is finite. Note that $-d^\top x^{(i)} = \min \partial f(\delta^{(i)})$. As $f(\delta^*) = 0$, the Bregman divergence of $\delta^{(i)}$ and δ^* can be written as

$$D_f(\delta^*, \delta^{(i)}) = f(\delta^{(i)}) + \max_{g \in \partial f(\delta^{(i)})} g(\delta^* - \delta^{(i)}) = (c - \delta^{(i)} d)^\top x^{(i)} - d^\top x^{(i)} (\delta^* - \delta^{(i)}) = (c - \delta^* d)^\top x^{(i)}.$$

According to Lemma 2.4, $(c - \delta^* d)^\top x^{(i)} = D_f(\delta^*, \delta^{(i)}) < \frac{1}{2} D_f(\delta^*, \delta^{(i-2)}) = \frac{1}{2} (c - \delta^* d)^\top x^{(i-2)}$ for all $3 \leq i \leq k$. By Lemma 2.3, we also know that $D_f(\delta^*, \delta^{(i)}) > 0$ for all $1 \leq i \leq k-2$. Thus, applying Lemma 2.5 yields $k = O(m \log m)$. ◀

2.2 Linear Fractional Programming

We next consider *linear fractional programming*, an extension of (1) with the assumption that the domain $\mathcal{D} \subseteq \mathbb{R}^m$ is a polyhedron, but removing the condition $d^\top x > 0$ for $x \in \mathcal{D}$. For $c, d \in \mathbb{R}^m$, the problem is

$$\inf c^\top x / d^\top x \quad \text{s.t.} \quad d^\top x > 0, \quad x \in \mathcal{D}. \quad (\text{F})$$

For the problem to be meaningful, we assume that $\mathcal{D} \cap \{x : d^\top x > 0\} \neq \emptyset$. The common form in the literature assumes $d^\top x > 0$ for all $x \in \mathcal{D}$ as in (1); we consider the more general setup for the purpose of solving M2VPI systems in Section 3. It is easy to see that any linear fractional combinatorial optimization problem on a domain $\mathcal{X} \subseteq \{0, 1\}^m$ can be cast as a linear fractional program with the polytope $\mathcal{D} = \text{conv}(\mathcal{X})$ because $c^\top \bar{x} / d^\top \bar{x} \geq \min_{x \in \mathcal{X}} c^\top x / d^\top x$ for all $\bar{x} \in \mathcal{D}$. The next theorem characterizes when (F) is unbounded.

► **Theorem 2.7.** *If $\mathcal{D} \cap \{x : d^\top x > 0\} \neq \emptyset$, then the optimal value of (F) is $-\infty$ if and only if at least one of the following two conditions hold:*

1. *There exists $x \in \mathcal{D}$ such that $c^\top x < 0$ and $d^\top x = 0$;*
2. *There exists $y \in \mathbb{R}^m$ such that $c^\top y < 0$, $d^\top y = 0$ and $x + \lambda y \in \mathcal{D}$ for all $x \in \mathcal{D}, \lambda \geq 0$.*

► **Example 2.8.** Unlike in linear programming, the optimal value may not be attained even if it is finite. Consider the instance given by $\inf(-x_1 + x_2)/(x_1 + x_2)$ subject to $x_1 + x_2 > 0$ and $-x_1 + x_2 = 1$. The numerator is equal to 1 for any feasible solution, while the denominator can be made arbitrarily large. Hence, the optimal value of this program is 0, which is not attained in the feasible region.

We use the Newton–Dinkelbach method for f as in (2), that is, $f(\delta) = \inf_{x \in \mathcal{D}} (c - \delta d)^\top x$. Since $\mathcal{D} \neq \emptyset$, $f(\delta) < \infty$ for all $\delta \in \mathbb{R}$. By the Minkowski–Weyl theorem, there exist finitely many points $P \subseteq \mathcal{D}$ such that $f(\delta) = \min_{x \in P} (c - \delta d)^\top x$ for all $\delta \in \text{dom}(f)$. Hence, f is concave and piecewise linear. Observe that $f(\delta) > -\infty$ if and only if every ray y in the recession cone of \mathcal{D} satisfies $(c - \delta d)^\top y \geq 0$. For f to be proper, we need to assume that Condition 2 in Theorem 2.7 does not hold. Moreover, we require the existence of a point $\delta' \in \text{dom}(f)$ such that $f(\delta') = (c - \delta' d)^\top x' \leq 0$ for some $x' \in \mathcal{D}$ with $d^\top x' > 0$. It follows that f has a root or attains its maximum because $\text{dom}(f)$ is closed. We are ready to characterize the optimal value of (F) using f .

► **Lemma 2.9.** *Assume that there exists $\delta' \in \text{dom}(f)$ such that $f(\delta') = (c - \delta' d)^\top x' \leq 0$ for some $x' \in \mathcal{D}$ with $d^\top x' > 0$. If f has a root, then the optimal value of (F) is equal to the largest root and is attained. Otherwise, the optimal value is $-\infty$.*

3 Monotone Two Variables per Inequality Systems

Recall that an M2VPI system can be represented as a directed multigraph $G = (V, E)$ with arcs costs $c \in \mathbb{R}^m$ and gain factors $\gamma \in \mathbb{R}_{++}^m$. For a u - v walk P in G with $E(P) = (e_1, e_2, \dots, e_k)$, its *cost* and *gain factor* are defined as $c(P) := \sum_{i=1}^k \left(\prod_{j=1}^{i-1} \gamma_{e_j} \right) c_{e_i}$ and $\gamma(P) := \prod_{i=1}^k \gamma_{e_i}$ respectively. If P is a single vertex, then $c(P) := 0$ and $\gamma(P) := 1$. The walk P induces the valid inequality $y_u \leq c(P) + \gamma(P)y_v$, implied by the sequence of arcs/inequalities in $E(P)$. It is also worth considering the dual interpretation. Dual variables on arcs correspond to generalized flows: if 1 unit of flow enter the arc $e = (u, v)$ at u , then γ_e units reach v , at a shipping cost of c_e . Thus, if 1 unit of flow enter a path P , then $\gamma(P)$ units reach the end of the path, incurring a cost of $c(P)$.

Given node labels $y \in \mathbb{R}^n$, the y -cost of a u - v walk P is defined as $c(P) + \gamma(P)y_v$. Note that the y -cost of a walk only depends on the label at the sink. A u - v path is called a *shortest u - v path with respect to y* if it has the smallest y -cost among all u - v walks. A *shortest path from u with respect to y* is a shortest u - v path with respect to y for some node v . Such a path does not always exist, as demonstrated in the full version.

If P is a u - u walk such that its intermediate nodes are distinct, then it is called a *cycle at u* . Given a u - v walk P and a v - w walk Q , we denote PQ as the u - w walk obtained by concatenating P and Q .

► **Definition 3.1.** A cycle C is called flow-generating if $\gamma(C) > 1$, unit-gain if $\gamma(C) = 1$, and flow-absorbing if $\gamma(C) < 1$. We say that a unit-gain cycle C is negative if $c(C) < 0$.

Note that $c(C)$ depends on the starting point u of a cycle C . This ambiguity is resolved by using the term *cycle at u* . For a unit-gain cycle C , it is not hard to see that the starting point does not affect the sign of $c(C)$. Hence, the definition of a negative unit-gain cycle is sound. Observe that a flow-absorbing cycle C induces an upper bound $y_u \leq \frac{c(C)}{1-\gamma(C)}$, while a flow-generating cycle C induces a lower bound $y_u \geq \frac{-c(C)}{\gamma(C)-1}$. Let $\mathcal{C}_u^{abs}(G)$ and $\mathcal{C}_u^{gen}(G)$ denote the set of flow-absorbing cycles and flow-generating cycles at u in G respectively.

► **Definition 3.2.** Given a flow-generating cycle C at u , a flow-absorbing cycle D at v , and a u - v path P , the walk CPD is called a bicycle. We say that the bicycle is negative if $c(P) + \gamma(P)\frac{c(D)}{1-\gamma(D)} < \frac{-c(C)}{\gamma(C)-1}$.

Using these two structures, Shostak characterized the feasibility of M2VPI systems.

► **Theorem 3.3** ([21]). An M2VPI system (G, c, γ) is infeasible if and only if G contains a negative unit-gain cycle or a negative bicycle.

3.1 A Linear Fractional Programming Formulation

Our goal is to compute the pointwise maximal solution $y^{\max} \in \bar{\mathbb{R}}^n$ to an M2VPI system if it is feasible, where $y_u^{\max} := \infty$ if and only if the variable y_u is unbounded from above. It is well known how to convert y^{\max} into a finite feasible solution – we refer to the full version for details. In order to apply Algorithm 1, we first need to reformulate the problem as a linear fractional program. Now, every coordinate y_u^{\max} can be expressed as the following primal-dual pair of linear programs, where $\nabla x_v := \sum_{e \in \delta^+(u)} x_e - \sum_{e \in \delta^-(u)} \gamma_e x_e$ denotes the net flow at a node v .

$$\begin{array}{ll} \min c^\top x & (P_u) \quad \max y_u \quad (D_u) \\ \text{s. t. } \nabla x_u = 1 & \text{s. t. } y_v - \gamma_e y_w \leq c_e \quad \forall e = (v, w) \in E \\ \nabla x_v = 0 \quad \forall v \in V \setminus u & \\ x \geq 0 & \end{array}$$

The primal LP (P_u) is a minimum-cost generalized flow problem with a supply of 1 at node u . It asks for the cheapest way to destroy one unit of flow at u . Observe that it is feasible if and only if u can reach a flow-absorbing cycle in G . If it is feasible, then it is unbounded if and only if there exists a negative unit-gain cycle or a negative bicycle in G . It can be reformulated as the following linear fractional program

$$\inf \frac{c^\top x}{1 - \sum_{e \in \delta^-(u)} \gamma_e x_e} \quad \text{s.t. } 1 - \sum_{e \in \delta^-(u)} \gamma_e x_e > 0, x \in \mathcal{D}. \quad (F_u)$$

with the polyhedron

$$\mathcal{D} := \{x \in \mathbb{R}_+^m : x(\delta^+(u)) = 1, \nabla x_v = 0 \forall v \in V \setminus u\}.$$

Indeed, if x is a feasible solution to (P_u) , then $x/x(\delta^+(u))$ is a feasible solution to (F_u) with the same objective value. This is because $1 - \sum_{e \in \delta^-(u)} \gamma_e x_e / x(\delta^+(u)) = 1/x(\delta^+(u))$. Conversely, if x is a feasible solution to (F_u) , then $x/(1 - \sum_{e \in \delta^-(u)} \gamma_e x_e)$ is a feasible solution to (P_u) with the same objective value. Even though the denominator is an affine function of x , it can be made linear to conform with (F) by working with the polyhedron $\{(x, 1) : x \in \mathcal{D}\}$.

Our goal is to solve (F_u) using Algorithm 1. For a fixed $\delta \in \mathbb{R}$, the value of the parametric function $f(\delta)$ can be written as the following pair of primal and dual LPs respectively

$$\begin{array}{ll}
\min & c^\top x + \delta \sum_{e \in \delta^-(u)} \gamma_e x_e - \delta \\
\text{s. t.} & x \in \mathcal{D} \\
\max & y_u - \delta \\
\text{s. t.} & y_v - \gamma_e \delta \leq c_e \quad \forall e = (v, u) \in \delta^-(u) \\
& y_v - \gamma_e y_w \leq c_e \quad \forall e = (v, w) \notin \delta^-(u).
\end{array}$$

We refer to them as the *primal (resp. dual) LP for $f(\delta)$* , and their corresponding feasible/optimal solution as a *feasible/optimal primal (resp. dual) solution to $f(\delta)$* .

Due to the specific structure of this linear fractional program, a suitable initial point for the Newton–Dinkelbach method can be obtained from any feasible solution to (F_u) . This is a consequence of the unboundedness test given by the following lemma.

► **Lemma 3.4.** *Let x be a feasible solution to (F_u) and $\bar{\delta} := c^\top x / (1 - \sum_{e \in \delta^-(u)} \gamma_e x_e)$. If either $f(\bar{\delta}) = -\infty$ or $f(\bar{\delta}) = c^\top \bar{x} - \bar{\delta}(1 - \sum_{e \in \delta^-(u)} \gamma_e \bar{x}_e) < 0$ for some $\bar{x} \in \mathcal{D}$ with $1 - \sum_{e \in \delta^-(u)} \gamma_e \bar{x}_e \leq 0$, then the optimal value of (F_u) is $-\infty$.*

In order to characterize the finiteness of $f(\delta)$, we introduce the following notion of a negative flow-generating cycle.

► **Definition 3.5.** *For a fixed $\delta \in \mathbb{R}$ and $u \in V$, a flow-generating cycle C is said to be (δ, u) -negative if there exists a path P from a node $v \in V(C)$ to node u such that $c(C) + (\gamma(C) - 1)(c(P) + \gamma(P)\delta) < 0$, where C is treated as a v - v walk in $c(C)$.*

► **Lemma 3.6.** *For any $\delta \in \mathbb{R}$, $f(\delta) = -\infty$ if and only if $\mathcal{D} \neq \emptyset$ and there exists a negative unit-gain cycle, a negative bicycle, or a (δ, u) -negative flow-generating cycle in $G \setminus \delta^+(u)$.*

It turns out that if we have an optimal dual solution y to $f(\delta)$ for some $\delta \in \mathbb{R}$, then we can compute an optimal dual solution to $f(\delta')$ for any $\delta' < \delta$. A suitable subroutine for this task is the so called GRAPEVINE algorithm (Algorithm 2), developed by Aspvall and Shiloach [2].

■ **Algorithm 2** GRAPEVINE.

input : A directed multigraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^m$ and gain factors $\gamma \in \mathbb{R}_{++}^m$, node labels $y \in \mathbb{R}^n$, and a node $u \in V$.
output : Node labels $y \in \mathbb{R}^n$ and a walk P of length at most n starting from u .

```

1 for  $i = 1$  to  $n$  do
2   foreach  $v \in V$  do
3      $y'_v \leftarrow \min(y_v, \min_{vw \in \delta^+(v)} c_{vw} + \gamma_{vw} y_w)$ 
4     if  $y'_v < y_v$  then
5        $\text{pred}(v, i) \leftarrow \arg \min_{vw \in \delta^+(v)} c_{vw} + \gamma_{vw} y_w$  /* Break ties */
6     else
7        $\text{pred}(v, i) \leftarrow \emptyset$ 
8    $y \leftarrow y'$ 
9 Let  $P$  be the walk obtained by tracing from  $\text{pred}(u, n)$ 
10 return  $(y, P)$ 

```

Given initial node labels $y \in \mathbb{R}^n$ and a specified node u , GRAPEVINE runs for n iterations. We say that an arc $e = (v, w)$ is *violated with respect to y* if $y_v > c_e + \gamma_e y_w$. In an iteration $i \in [n]$, the algorithm records the most violated arc with respect to y in $\delta^+(v)$ as $\text{pred}(v, i)$, for each node $v \in V$ (ties are broken arbitrarily). Note that $\text{pred}(v, i) = \emptyset$ if there are

no violated arcs in $\delta^+(v)$. Then, each y_v is decreased by the amount of violation in the corresponding recorded arc. After n iterations, the algorithm traces a walk P from u by following the recorded arcs in reverse chronological order. During the trace, if $\text{pred}(v, i) = \emptyset$ for some $v \in V$ and $i > 1$, then $\text{pred}(v, i - 1)$ is read. Finally, the updated node labels y and the walk P are returned. Clearly, the running time of GRAPEVINE is $O(mn)$.

Given an optimal dual solution $y \in \mathbb{R}^n$ to $f(\delta)$ and $\delta' < \delta$, the dual LP for $f(\delta')$ can be solved using GRAPEVINE as follows. Define the directed graph $G_u := (V \cup \{u'\}, E_u)$ where $E_u := (E \setminus \delta^-(u)) \cup \{vu' : vu \in \delta^-(u)\}$. The graph G_u is obtained from G by splitting u into two nodes u, u' and reassigning the incoming arcs of u to u' . These arcs inherit the same costs and gain factors from their counterparts in G . Let $\bar{y} \in \mathbb{R}^{n+1}$ be node labels in G_u defined by $\bar{y}_{u'} := \delta'$ and $\bar{y}_v := y_v$ for all $v \neq u'$. Then, we run GRAPEVINE on G_u with input node labels \bar{y} and node u . Note that $\bar{y}_{u'}$ remains unchanged throughout the algorithm. The next lemma verifies the correctness of this method.

► **Lemma 3.7.** *Given an optimal dual solution $y \in \mathbb{R}^n$ to $f(\delta)$ and $\delta' < \delta$, define $\bar{y} \in \mathbb{R}^{n+1}$ as $\bar{y}_{u'} := \delta'$ and $\bar{y}_v := y_v$ for all $v \in V$. Let (\bar{z}, P) be the node labels and walk returned by $\text{GRAPEVINE}(G_u, \bar{y}, u)$. If \bar{z}_V is not feasible to the dual LP for $f(\delta')$, then $f(\delta') = -\infty$. Otherwise, \bar{z}_V is a dual optimal solution to $f(\delta')$ and P is a shortest path from u with respect to \bar{y} in G_u .*

If \bar{z}_V is an optimal dual solution to $f(\delta')$, a supergradient in $\partial f(\delta')$ can be inferred from the returned path P . We say that an arc $e = (v, w)$ is *tight with respect to \bar{z}* if $\bar{z}_v = c_e + \gamma_e \bar{z}_w$. By complementary slackness, every optimal primal solution to $f(\delta')$ is supported on the subgraph of G_u induced by tight arcs with respect to \bar{z} . In particular, any u - u' path or any path from u to a flow-absorbing cycle in this subgraph constitutes a basic optimal primal solution to $f(\delta')$. As P is also a path in this subgraph, we have $\gamma(P) - 1 \in \partial f(\delta')$ if P ends at u' . Otherwise, u can reach a flow-absorbing cycle in this subgraph because $\delta' < \delta$. In this case, $-1 \in \partial f(\delta')$.

3.2 A Strongly Polynomial Label-Correcting Algorithm

Using Algorithm 1, we develop a strongly polynomial label-correcting algorithm for solving an M2VPI system (G, c, γ) . The main idea is to start with a subsystem for which (D_u) is trivial, and progressively solve (D_u) for larger and larger subsystems. Throughout the algorithm, we maintain node labels $y \in \mathbb{R}^n$ which form valid upper bounds on each variable. They are initialized to ∞ at every node. We also maintain a subgraph of G , which initially is $G^{(0)} := (V, \emptyset)$.

The algorithm (Algorithm 3) is divided into n phases. At the start of phase $k \in [n]$, a new node $u \in V$ is selected and all of its outgoing arcs in G are added to $G^{(k-1)}$, resulting in a larger subgraph $G^{(k)}$. Since $y_u = \infty$ at this point, we update it to the smallest upper bound implied by its outgoing arcs and the labels of its outneighbours. If y_u is still infinity, then we know that $\delta^+(u) = \emptyset$ or $y_v = \infty$ for all $v \in N^+(u)$. In this case, we find a flow-absorbing cycle at u in $G^{(k)}$ using the multiplicative Bellman–Ford algorithm, by treating the gain factors as arc costs. If there is none, then we proceed to the next phase immediately as y_u is unbounded from above in the subsystem $(G^{(k)}, c, \gamma)$. This is because u cannot reach a flow-absorbing cycle in $G^{(k)}$ by induction. We would like to point out that this does not necessarily imply that the full system (G, c, γ) is feasible (see the full version for details). On the other hand, if Bellman–Ford returns a flow-absorbing cycle, then y_u is set to the upper bound implied by the cycle. Then, we apply Algorithm 1 to solve (D_u) for the subsystem $(G^{(k)}, c, \gamma)$.

■ **Algorithm 3** Label-correcting algorithm for M2VPI systems.

input : An M2VPI system (G, c, γ) .
output : The pointwise maximal solution y^{\max} or the string INFEASIBLE.

```

1 Initialize graph  $G^{(0)} \leftarrow (V, \emptyset)$  and counter  $k \leftarrow 0$ 
2 Initialize node labels  $y \in \bar{\mathbb{R}}^n$  as  $y_v \leftarrow \infty \forall v \in V$ 
3 foreach  $u \in V$  do
4    $k \leftarrow k + 1$ 
5    $G^{(k)} \leftarrow G^{(k-1)} \cup \delta^+(u)$ 
6    $y_u \leftarrow \min_{uv \in \delta^+(u)} c_{uv} + \gamma_{uv} y_v$ 
7   if  $y_u = \infty$  and  $\mathcal{C}_u^{\text{abs}}(G^{(k)}) \neq \emptyset$  then
8      $y_u \leftarrow c(C)/(1 - \gamma(C))$  for any  $C \in \mathcal{C}_u^{\text{abs}}(G^{(k)})$ 
9   if  $y_u < \infty$  then
10    Define node labels  $\bar{y} \in \bar{\mathbb{R}}^{n+1}$  as  $\bar{y}_{u'} \leftarrow y_u$  and  $\bar{y}_v \leftarrow y_v \forall v \in V$ 
11     $(\bar{y}, P) \leftarrow \text{GRAPEVINE}(G_u^{(k)}, \bar{y}, u)$ 
12    if  $\exists$  a violated arc w.r.t.  $\bar{y}$  in  $G_u^{(k)}$  or  $(|E(P)| > 0 \text{ and } \gamma(P) \geq 1)$  then
13      return INFEASIBLE
14     $\bar{y}_{u'} \leftarrow \text{LOOK-AHEAD-NEWTON}(\text{GRAPEVINE}(G_u^{(k)}, \cdot, u), \bar{y}_{u'}, \gamma(P) - 1)$ 
15    if  $\bar{y}_{u'} = \text{NO ROOT}$  then
16      return INFEASIBLE
17     $y \leftarrow \bar{y}_V$ 
18 return  $y$ 

```

The value and supergradient oracle for the parametric function $f(\delta)$ is GRAPEVINE. Let $G_u^{(k)}$ be the modified graph and $\bar{y} \in \bar{\mathbb{R}}^{n+1}$ be the node labels as defined in the previous subsection. In order to provide Algorithm 1 with a suitable initial point and supergradient, we run GRAPEVINE on $G_u^{(k)}$ with input node labels \bar{y} . It updates \bar{y} and returns a walk P from u . If \bar{y}_V is not feasible to the dual LP for $f(\bar{y}_{u'})$ or P is a non-trivial walk with $\gamma(P) \geq 1$, then we declare infeasibility. Otherwise, we run Algorithm 1 with the initial point $\bar{y}_{u'}$ and supergradient $\gamma(P) - 1$. We remark that GRAPEVINE continues to update \bar{y} throughout the execution of Algorithm 1.

► **Theorem 3.8.** *If Algorithm 3 returns $y \in \bar{\mathbb{R}}^n$, then $y = y^{\max}$ if the M2VPI system is feasible. Otherwise, the system is infeasible.*

We would like to point out that Algorithm 3 may return node labels $y \in \bar{\mathbb{R}}^n$ even if the M2VPI system is infeasible. This happens when y contains ∞ entries. It is well-known how to ascertain the system's feasibility status in this case. We refer the reader to the full version for details.

To bound the running time of Algorithm 3, it suffices to bound the running time of Algorithm 1 in every phase. Our strategy is to analyze the sequence of paths whose gain factors determine the right derivative of f at each iterate of Algorithm 1. The next property is crucial in our arc elimination argument.

► **Definition 3.9.** *Let $\mathcal{P} = (P^{(1)}, P^{(2)}, \dots, P^{(\ell)})$ be a sequence of paths from u . We say that \mathcal{P} satisfies subpath monotonicity at u if for every pair $P^{(i)}, P^{(j)}$ where $i < j$ and for every shared node $v \neq u$, we have $\gamma(P_{uv}^{(i)}) \leq \gamma(P_{uv}^{(j)})$.*

► **Lemma 3.10.** *Let $\delta^{(1)} > \delta^{(2)} > \dots > \delta^{(\ell)}$ be a decreasing sequence of iterates. For each $\delta^{(i)} \in \mathbb{R}$, let $P^{(i)}$ be a u - u' path in G_u on which a unit flow is an optimal primal solution to $f(\delta^{(i)})$. Then, the sequence $(P^{(1)}, P^{(2)}, \dots, P^{(\ell)})$ satisfies subpath monotonicity at u .*

Proof. For each $i \in [\ell]$, let $y^{(i)} \in \mathbb{R}^n$ be an optimal dual solution to $f(\delta^{(i)})$. Let $\bar{y}^{(i)} \in \mathbb{R}^{n+1}$ be the node labels in G_u defined by $\bar{y}_{u'}^{(i)} := \delta^{(i)}$ and $\bar{y}_v := y_v$ for all $v \neq u'$. By complementary slackness, every arc in $P^{(i)}$ is tight with respect to $\bar{y}^{(i)}$. Hence, $P^{(i)}$ is a shortest u - u' path in G_u with respect to $\bar{y}^{(i)}$. Now, pick a pair of paths $P^{(i)}$ and $P^{(j)}$ such that $i < j$ and they share a node $v \neq u$. Then, the subpaths $P_{uv}^{(i)}$ and $P_{uv}^{(j)}$ are also shortest u - v paths in G_u with respect to $\bar{y}^{(i)}$ and $\bar{y}^{(j)}$ respectively. Observe that $\bar{y}_v^{(i)} > \bar{y}_v^{(j)}$ because $\bar{y}_{u'}^{(i)} = \delta^{(i)} > \delta^{(j)} = \bar{y}_{u'}^{(j)}$. Define the function $\psi : [\bar{y}_v^{(j)}, \bar{y}_v^{(i)}] \rightarrow \mathbb{R}$ as

$$\psi(x) := \inf \{c(P) + \gamma(P)x : P \text{ is a } u\text{-}v \text{ walk in } G_u\}.$$

Clearly, it is increasing and concave. It is also finite because $\psi(\bar{y}_v^{(i)}) = c(P_{uv}^{(i)}) + \gamma(P_{uv}^{(i)})\bar{y}_v^{(i)}$ and $\psi(\bar{y}_v^{(j)}) = c(P_{uv}^{(j)}) + \gamma(P_{uv}^{(j)})\bar{y}_v^{(j)}$. Subpath monotonicity then follows from the concavity of ψ . ◀

► **Theorem 3.11.** *In each phase k of Algorithm 3, Algorithm 1 terminates in $O(|E(G^{(k)})|)$ iterations.*

The full proof is given in the full version; we highlight some key steps. Let $m_k := |E(G^{(k)})|$. Let $\bar{y} = (\bar{y}^{(1)}, \bar{y}^{(2)}, \dots, \bar{y}^{(\ell)})$ be a sequence of node labels at the start of every iteration of Algorithm 1 in phase k . Let $\mathcal{P} = (P^{(2)}, P^{(3)}, \dots, P^{(\ell)})$ be a sequence of u - u' paths in $G_u^{(k)}$ such that $P^{(i)}$ determines the right derivative $f'_+(\bar{y}_{u'}^{(i)})$ for all $i > 1$. Perturb $\hat{c} := c + \varepsilon \chi_{\delta^{(u)}}$ by a suitably small $\varepsilon \geq 0$ such that the system $(G^{(k)}, \hat{c}, \gamma)$ is feasible. Let $y^* \in \mathbb{R}^n$ be its pointwise maximal solution, and define the reduced cost $c^* \in \mathbb{R}_+^{m_k}$ as $c_{vw}^* := \hat{c}_{vw} + \gamma_{vw}y_{vw}^* - y_v^*$ for all $vw \in E(G^{(k)})$. For every arc $vw \in \cup_{i=1}^{\ell} E(P^{(i)})$, let r_{vw} be the largest gain factor of the u - v subpath of the paths in \mathcal{P} that contain vw . By Lemma 3.10, this is achieved by the last path in \mathcal{P} which contains vw . Order the elements of $r \circ c^*$ as $0 \leq r_1 c_1^* \leq r_2 c_2^* \leq \dots \leq r_{m_k} c_{m_k}^*$, and let $d_i := \sum_{j=1}^i r_j c_j^*$ for every $i \in [m_k]$. The final step is showing that every interval $(d_i, d_{i+1}]$ contains the cost of at most two paths from \mathcal{P} . This can be derived from the Bregman divergence analysis, that yields $c^*(P^{(i)}) \leq \frac{1}{2}c^*(P^{(i-2)})$ for all $3 \leq i \leq \ell$.

The runtime of every iteration of Algorithm 1 is dominated by GRAPEVINE. Thus, we obtain the following result.

► **Corollary 3.12.** *Algorithm 3 solves the feasibility of M2VPI linear systems in $O(m^2 n^2)$ time.*

References

- 1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, 1993.
- 2 Bengt Aspvall and Yossi Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9(4):827–845, 1980.
- 3 Edith Cohen and Nimrod Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23(6):1313–1347, 1994.
- 4 Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.
- 5 Herbert Edelsbrunner, Günter Rote, and Emo Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theor. Comput. Sci.*, 66(2):157–180, 1989.

- 6 Eugene A. Feinberg and Jefferson Huang. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. *Oper. Res. Lett.*, 42(2):130–131, 2014.
- 7 Michel X. Goemans, Swati Gupta, and Patrick Jaillet. Discrete Newton’s algorithm for parametric submodular function minimization. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization*, pages 212–227, 2017.
- 8 Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.
- 9 Thomas Dueholm Hansen, Haim Kaplan, and Uri Zwick. Dantzig’s pivoting rule for shortest paths, deterministic MDPs, and minimum cost to time ratio cycles. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 847–860, 2014.
- 10 Dorit S. Hochbaum, Nimrod Megiddo, Joseph Naor, and Arie Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Math. Program.*, 62:69–83, 1993.
- 11 Dorit S. Hochbaum and Joseph Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.
- 12 Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence*, pages 394–402, 1995.
- 13 Omid Madani. On policy iteration as a Newton’s method and polynomial policy iteration algorithms. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 273–278, 2002.
- 14 Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.
- 15 Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12(2):347–353, 1983.
- 16 Neil Olver and László A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *J. ACM*, 67(2):10:1–10:26, 2020.
- 17 Ian Post and Yinyu Ye. The simplex method is strongly polynomial for deterministic Markov decision processes. *Math. Oper. Res.*, 40(4):859–868, 2015.
- 18 Tomasz Radzik. Newton’s method for fractional combinatorial optimization. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992.
- 19 Tomasz Radzik. Fractional combinatorial optimization. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume 1–3*, pages 429–478. Springer US, 1998.
- 20 Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, 1994.
- 21 Robert E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981.
- 22 Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998.
- 23 Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.
- 24 László A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.*, 42(1):179–211, 2017.
- 25 Qin Wang, Xiaoguang Yang, and Jianzhong Zhang. A class of inverse dominant problems under weighted ℓ_∞ norm and an improved complexity bound for Radzik’s algorithm. *J. Global Optimization*, 34(4):551–567, 2006.


Faster 3-Coloring of Small-Diameter Graphs

Michał Dębski ✉

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Marta Piecyk ✉

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Paweł Rzażewski ✉ 

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Abstract

We study the 3-COLORING problem in graphs with small diameter. In 2013, Mertzios and Spirakis showed that for n -vertex diameter-2 graphs this problem can be solved in subexponential time $2^{\mathcal{O}(\sqrt{n \log n})}$. Whether the problem can be solved in polynomial time remains a well-known open question in the area of algorithmic graphs theory.

In this paper we present an algorithm that solves 3-COLORING in n -vertex diameter-2 graphs in time $2^{\mathcal{O}(n^{1/3} \log^2 n)}$. This is the first improvement upon the algorithm of Mertzios and Spirakis in the general case, i.e., without putting any further restrictions on the instance graph.

In addition to standard branchings and reducing the problem to an instance of 2-SAT, the crucial building block of our algorithm is a combinatorial observation about 3-colorable diameter-2 graphs, which is proven using a probabilistic argument.

As a side result, we show that 3-COLORING can be solved in time $2^{\mathcal{O}((n \log n)^{2/3})}$ in n -vertex diameter-3 graphs. We also generalize our algorithms to the problem of finding a list homomorphism from a small-diameter graph to a cycle.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases 3-coloring, fine-grained complexity, subexponential-time algorithm, diameter

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.37

Funding Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

Acknowledgements The authors are sincerely grateful to Carla Groenland for many inspiring discussions and useful comments on the manuscript.

1 Introduction

For many NP-hard graph problems, the instances constructed in hardness reductions are very specific and “unstructured”. Thus a natural direction of research is to study how additional restrictions imposed on the input graphs affect the complexity of the problem. In particular, we would like to understand if the additional knowledge about the structure of the instance makes the problem easier, and what are the “minimal” sets of restrictions that we need to impose in order to make the problem efficiently solvable.

Usually, the main focus in the area is on *hereditary* classes of graphs, i.e., classes that are closed under vertex deletion. Prominent examples are perfect graphs [7, 18], graphs excluding a certain induced subgraph [17] or minor [11], and intersection graphs of geometric objects [19]. Studying these classes has led to a better understanding of the structure of such graphs [8, 9, 20, 29] and a discovery of numerous exciting algorithmic techniques [2, 10, 15, 16, 24]. Let us point out that the property of being hereditary is particularly useful in the construction of recursive algorithms based on branching or the divide & conquer paradigm.



© Michał Dębski, Marta Piecyk, and Paweł Rzażewski;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, 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

However, there are many natural classes of graphs that are not hereditary, for example graphs with bounded diameter. Such graphs are interesting not only for purely theoretical reasons: for example social networks tend to have small diameter [30].

Observe that for any graph G , a graph G^+ obtained from G by adding a universal vertex has diameter 2. Since the graph G may be arbitrarily complicated, the fact that G^+ has small diameter does not imply that its structure is simple. This observation can be used to show that many classic computational problems are NP-hard for graphs of bounded diameter and they cannot be solved in subexponential time under the ETH. For instance, the size of a maximum independent set in G^+ is equal to the size of a maximum independent set in G , and thus MAX INDEPENDENT SET in diameter-2 graph is NP-hard and cannot be solved in subexponential time, unless the ETH fails.

A similar argument applies to k -COLORING: the graph G^+ is k -colorable if and only if G is $(k - 1)$ -colorable. Thus, for any $k \geq 4$, the k -COLORING problem is NP-hard and admits no subexponential-time algorithm (under the ETH) in diameter-2 graphs. However, the reasoning above breaks down for $k = 3$, as 2-COLORING is polynomial-time solvable.

This peculiar open case was first studied by Mertzios, Spirakis [26] who proved that the problem can be solved in *subexponential time*. The result holds even for the more general LIST 3-COLORING problem, where each vertex v of the instance graph is equipped with a *list* $L(v) \subseteq \{1, 2, 3\}$, and we ask for a proper coloring, in which every vertex gets a color from its list.

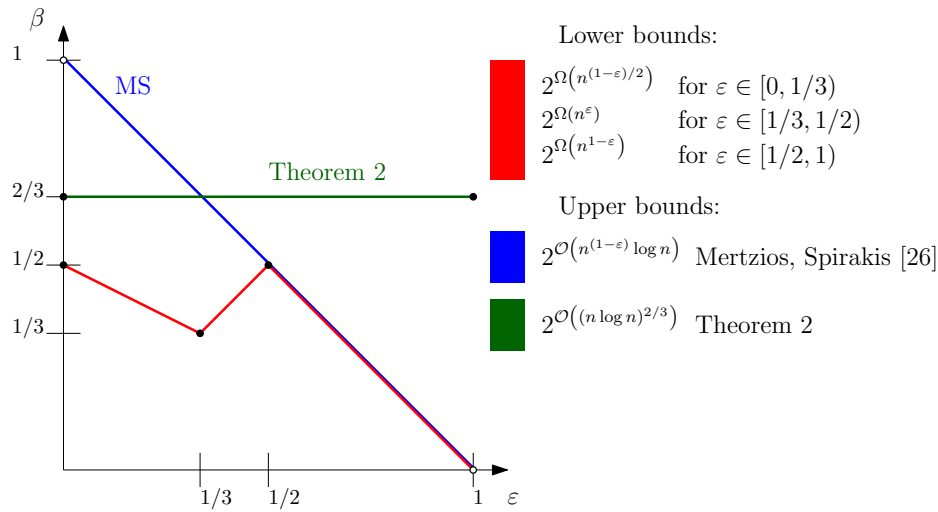
► **Theorem 1** (Mertzios, Spirakis [26]). *The LIST 3-COLORING problem on n -vertex graphs with diameter 2 can be solved in time $2^{\mathcal{O}(\sqrt{n \cdot \log n})}$.*

Their algorithm is based on a simple win-win argument. The first ingredient is a well-known fact that every graph with n vertices and minimum degree δ has a dominating set of size $\mathcal{O}\left(\frac{n \log \delta}{\delta}\right)$ [1, Theorem 1.2.2]. On the other hand, in a diameter-2 graph, the neighborhood of each vertex is a dominating set, so there is a dominating set of size δ . Thus, every diameter-2 graph has a dominating set S of size $\mathcal{O}\left(\min(\delta, \frac{n \log \delta}{\delta})\right)$ which is upper-bounded by $\mathcal{O}(\sqrt{n \log n})$.

We exhaustively guess the coloring of vertices in S and update the lists of their neighbors. Note that after this, each uncolored vertex has at least one colored neighbor, and thus each list has at most 2 elements. A classic result by Edwards [12] shows that such a problem can be solved in polynomial time by a reduction to 2-SAT. Summing up, the complexity of the algorithm is bounded by $2^{|S|} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(\sqrt{n \log n})}$.

Let us point out that the bound \sqrt{n} appears naturally for different parameters of diameter-2 graphs, for example the maximum degree of such a graph is $\Omega(\sqrt{n})$. Based on this, one can also construct different algorithms for LIST 3-COLORING in diameter-2 graphs with running time matching the one of Theorem 1 (see Section 3).

If it comes to 3-COLORING in diameter-3 graphs, Mertzios and Spirakis [26] proved that the problem is NP-hard, but their reduction is quadratic. Thus, under the ETH, the problem cannot be solved in time $2^{o(\sqrt{n})}$. Actually, the authors carefully analyzed how the lower bound depends on the minimum degree of the input graph, and presented three hardness reductions, each for a different range of δ . Furthermore, they showed that the problem can be solved in time $2^{\mathcal{O}(\min(\delta \cdot \Delta, \frac{n \log \delta}{\delta}))}$, where Δ is the maximum degree. The argument again follows from the observation that each diameter-3 graph has a dominating set of size at most $\delta \cdot \Delta$. Let us point out that if $\Delta = \Theta(n)$ and $\delta = \mathcal{O}(1)$, then the running time is exponential in n . In Figure 1 we summarize the results for diameter-3 graphs with given minimum degree.



■ **Figure 1** The complexity of LIST 3-COLORING in n -vertex diameter-3 graphs with minimum degree $\Theta(n^\varepsilon)$ for $\varepsilon \in [0, 1]$. The complexity bound is of the form $2^{\mathcal{O}(n^\beta \cdot \log^{\mathcal{O}(1)} n)}$ for $\beta \in [0, 1]$.

The story stops at diameter 3: a textbook reduction from NAE-SAT to 3-COLORING builds a graph with diameter 4 and number of vertices linear in the size of the formula [27, Theorem 9.8]. This proves that the 3-COLORING problem in diameter-4 graphs is NP-hard and cannot be solved in subexponential time, unless the ETH fails.

Closing the gaps left by Mertzios and Spirakis [26], and in particular determining the complexity of 3-COLORING in diameter-2 graphs, is a notorious open problem in the area of graph algorithms. We know polynomial-time algorithms if some additional restrictions are imposed on the instance [21, 23]. However, to the best of our knowledge, no progress in the general case has been achieved.

Let us also point out that some other problems, including different variants of graph coloring, have also been studied for small-diameter graphs [3, 5, 6, 22].

Our results. As our first result, in Section 3 we show a simple subexponential-time algorithm for the LIST 3-COLORING problem in diameter-3 graphs.

► **Theorem 2.** *The LIST 3-COLORING problem on n -vertex graphs with diameter 3 can be solved in time $2^{\mathcal{O}(n^{2/3} \cdot \log^{2/3} n)}$.*

Note that the running time bound does not depend on the maximum nor the minimum degree of the input graph. In particular, this is the *first* algorithm for LIST 3-COLORING, whose complexity is subexponential for *all* diameter-3 graphs, see Figure 1.

Let us present a high-level overview of the proof. We partition the vertex set of our graph into three sets V_1, V_2, V_3 , where V_i contains the vertices with lists of size i . If the graph contains a vertex $v \in V_2 \cup V_3$ with at least $n^{1/3}$ neighbors in $V_2 \cup V_3$, then we can effectively branch on the color of v . Otherwise, we observe that for any $v \in V_2 \cup V_3$, the set S of vertices at distance at most 2 from v in the graph induced by sets $V_2 \cup V_3$ dominates V_3 , i.e., every vertex from V_3 is in S or has a neighbor in S . Thus, after exhaustively guessing the coloring of S , all lists are reduced to size at most 2 and then we can finish in polynomial time, using the already-mentioned result of Edwards [12].

In Section 4 we prove the following theorem, which is the main result of the paper.

► **Theorem 3.** *The LIST 3-COLORING problem on n -vertex graphs with diameter 2 can be solved in time $2^{\mathcal{O}(n^{1/3} \cdot \log^2 n)}$.*

Again, let us give some intuition about the proof. We partition the vertex set of G into (V_1, V_2, V_3) , as previously. We aim to empty the set V_3 , as then the problem can be solved in polynomial time. We start with applying three branching rules. The first one is similar as in the proof of Theorem 2: if we find a vertex v with many neighbors in V_3 , we can branch on choosing the color of v . The other two branching rules are somewhat technical and their purpose is not immediately clear, so let us not discuss them here.

The main combinatorial insight that is used in our algorithm is as follows. Consider an instance (G, L) , where G is of diameter 2 and none of the previous branching rules can be applied. Suppose that G has a proper 3-coloring φ that respects lists L . Then there is a color $a \in \{1, 2, 3\}$ and sets $S \subseteq V_3 \cap \varphi^{-1}(a)$ and $\tilde{S} \subseteq V_3 \setminus \varphi^{-1}(a)$, each of size $\mathcal{O}(n^{1/3} \log n)$, with the following property:

(★) $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$ dominates at least $\frac{1}{6}$ -fraction of V_3 ,

where $N(S)$ (resp. $N(\tilde{S})$) denotes the set of vertices with a neighbor in S (resp. \tilde{S}). The existence of the sets S and \tilde{S} is shown using a probabilistic argument.

Now we proceed as follows. We enumerate all pairs of disjoint sets S and \tilde{S} , each of size $\mathcal{O}(n^{1/3} \log n)$. If they satisfy the property (★), we exhaustively guess the color a used for every vertex of S and the coloring of \tilde{S} with colors $\{1, 2, 3\} \setminus \{a\}$. Then we update the lists of the neighbors of colored vertices. Note that the color of every vertex from $N(S) \cap N(\tilde{S})$ is now uniquely determined. Thus, for at least $\frac{1}{6}$ -fraction of vertices $v \in V_3$, they are either already colored or have a colored neighbor and thus their lists are of size at most 2. Thus our instance was significantly simplified and we can proceed recursively.

Finally, in Section 5 we investigate possible extensions of our algorithms to some generalizations of (LIST) 3-COLORING. We observe that our approach can be used to obtain subexponential-time algorithms for the problem of finding a *list homomorphism* from a graph with diameter at most 3 to certain graphs, including in particular all cycles. We refer to Section 5.1 for the definition of the problem and the precise statement of our results; let us just point out that under the ETH the problems considered there cannot be solved in subexponential time in general graphs [13, 14]

We conclude with discussing the possibility of extending our algorithms to weighted coloring problems, with INDEPENDENT ODD CYCLE TRANSVERSAL [4] as a prominent special case.

2 Preliminaries

For an integer n , we denote $[n] := \{1, 2, \dots, n\}$. For a set X , by 2^X we denote the family of all subsets of X . All logarithms in the paper are natural.

Let $G = (V, E)$ be a connected graph. For two vertices u and v , by $\text{dist}_G(u, v)$ we denote the distance from u to v , i.e., the number of edges on a shortest u - v path in G . The *diameter* of G , denoted by $\text{diam}(G)$, is the maximum value of $\text{dist}(u, v)$ over all $u, v \in V$.

For a vertex v , by $N_G(v)$ we denote its *open neighborhood*, i.e., the set of all vertices adjacent to v . The *closed neighborhood* of v is defined as $N_G[v] := N_G(v) \cup \{v\}$. For an integer p , by $N_G^{\leq p}[v]$ we denote the set of vertices at distance at most p from v , and define $N_G^{\leq p}(v) := N_G^{\leq p}[v] \setminus \{v\}$. For a set X of vertices, we define $N_G(X) := \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] := N_G(X) \cup X$. For sets $X, Y \subseteq V$, we say that X *dominates* Y if $Y \subseteq N_G[X]$. By $\deg_G(v)$ we denote the *degree* of a vertex v , i.e., $|N_G(v)|$.

If the graph G is clear from the context, we drop the subscript in the notation above and simply write $\text{dist}(u, v)$, $N(v)$, etc. By $\Delta(G)$ we denote the maximum vertex degree in G .

The following result by Edwards [12] will be an important tool used in all our algorithms.

► **Theorem 4** (Edwards [12]). *Let $G = (V, E)$ be a graph and let $L : V \rightarrow 2^{\mathbb{N}}$ be a list assignment, such that for every $v \in V$ it holds that $|L(v)| \leq 2$. Then in polynomial time we can decide whether G admits a proper vertex coloring that respects lists L .*

Reduction rules. Let (G, L) be an instance of the LIST 3-COLORING problem. It is straightforward to observe that the following reduction rules can be safely applied, as they do not change the set of solutions. Moreover, each of them can be applied in polynomial time.

R1 If there exists a vertex v such that $L(v)$ contains only one color a , then remove a from $L(u)$ for each vertex $u \in N(v)$.

R2 If there exists a vertex v such that $|L(v)| = 0$, then report failure.

R3 If $|L(v)| \leq 2$ for each vertex v , then solve the problem using Theorem 4.

An instance (G, L) for which none of the reduction rules can be applied is called *reduced*. Note that the reduction rules do not remove any vertices from the graph, even if their color is fixed. This is because such an operation might increase the diameter.

Layer structure. Let (G, L) be a reduced instance of LIST 3-COLORING. For $i \in [3]$, let V_i be the set of vertices v of G , such that $|L(v)| = i$. Note that (V_1, V_2, V_3) is a partition of V ; we will call it the *layer structure* of G . Observe that since R1 cannot be applied to (G, L) , it holds that $N(V_1) \subseteq V_2$, i.e., there are no edges between V_1 and V_3 .

We conclude this section with an important observation about layer structures of graphs with diameter at most 3.

► **Proposition 5.** *Let (G, L) be a reduced instance of LIST 3-COLORING, where G has diameter $d \leq 3$, and let (V_1, V_2, V_3) be the layer structure of G . Then, for any $u, v \in V_2 \cup V_3$, at least one of the following holds:*

- a) u and v are at distance at most d in $G[V_2 \cup V_3]$, or
- b) $\{u, v\} \cap V_2 \neq \emptyset$.

Proof. If $V_1 = \emptyset$, then the first outcome follows, since $G = G[V_2 \cup V_3]$. So assume that $V_1 \neq \emptyset$. Consider $u, v \in V_3$ and suppose that they are not at distance at most d in $G[V_2 \cup V_3]$. Since they are at distance at most d in G , all shortest u - v -paths in G must intersect V_1 . However, for any $x \in V_1$, it holds that $\text{dist}(u, x) \geq 2$ and $\text{dist}(v, x) \geq 2$. Thus $\text{dist}(u, v) \geq 4$, contradicting the fact that $\text{diam}(G) \leq 3$. ◀

Observe that Proposition 5 does not generalize to diameter-4 graphs: consider e.g. 5-vertex path P_5 with consecutive vertices v_1, v_2, v_3, v_4, v_5 , where $V_1 = \{v_3\}$. Vertices v_1 and v_5 are in V_3 , they are at distance 4 in P_5 , but not in $P_5[V_2 \cup V_3] = P_5 - \{v_3\}$. For $d = 2$ Proposition 5 can be strengthened by replacing part (b) with $\{u, v\} \subseteq V_2$

Proposition 5 immediately yields the following corollary.

► **Corollary 6.** *Let (G, L) be an instance of the LIST 3-COLORING, where G has diameter $d \in \{2, 3\}$, and let (V_1, V_2, V_3) be the layer structure of G . For every $v \in V_3$, the set $N_{G[V_2 \cup V_3]}^{\leq d-1}[v]$ dominates V_3 .*

3 Coloring diameter-3 graphs

In this section we present a simple proof of Theorem 2. Actually, we will show the following more general result, which yields a $2^{\mathcal{O}(\sqrt{n \log n})}$ -algorithm for diameter-2 graphs; it implies Theorem 1, but the proof is different than the one originally given by Mertzios and Spirakis [26]. This will serve as a warm-up before showing our main result, i.e., Theorem 3.

► **Theorem 7.** *The LIST 3-COLORING problem on n -vertex graphs G can be solved in time:*

1. $2^{\mathcal{O}(n^{1/2} \log^{1/2} n)}$, if $\text{diam}(G) = 2$,
2. $2^{\mathcal{O}(n^{2/3} \log^{2/3} n)}$, if $\text{diam}(G) = 3$.

Proof. Let (G, L) be an instance of LIST 3-COLORING, where G has n vertices and diameter $d \in \{2, 3\}$. Without loss of generality we may assume that it is reduced. Let (V_1, V_2, V_3) be the layer structure of (G, L) and let us define a measure $\mu := 2|V_2| + 3|V_3|$.

First, consider the case that there is a vertex $v \in V_2 \cup V_3$ with at least $(\mu \log \mu)^{1/d}$ neighbors in $V_2 \cup V_3$. Since each vertex of $V_2 \cup V_3$ has one of four possible lists, there is a subset of at least $\frac{(\mu \log \mu)^{1/d}}{4}$ neighbors of v that all have the same list L' . Note that there is $a \in L(v) \cap L'$ since both are subsets of size at least 2 of a set of size 3. We branch on coloring the vertex v with color a or not. In other words, in the first branch we remove from $L(v)$ all elements but a , and in the other one we remove a from $L(v)$. Note that after reducing the obtained instance, at least $\frac{(\mu \log \mu)^{1/d}}{4}$ vertices will lose at least one element from their list in the first branch.

We can bound the number of instances produced by applying this step exhaustively as follows:

$$F(\mu) \leq F\left(\mu - \frac{(\mu \log \mu)^{1/d}}{4}\right) + F(\mu - 1).$$

Solving this inequality, we obtain that $F(\mu) = \mu^{\mathcal{O}\left(\frac{\mu}{(\mu \log \mu)^{1/d}}\right)} = 2^{\mathcal{O}((\mu \log \mu)^{1-1/d})}$.

Now consider the remaining case that $\Delta(G[V_2 \cup V_3]) < (\mu \log \mu)^{1/d}$. Recall that since the reduction rule R3 cannot be applied, it holds that $V_3 \neq \emptyset$. Pick any vertex $v \in V_3$. Define $X := N_{G[V_2 \cup V_3]}^{\leq d-1}[v]$; by Corollary 6, the set X dominates V_3 . Furthermore

$$|X| \leq 1 + \Delta(G[V_2 \cup V_3])^{d-1} = \mathcal{O}((\mu \log \mu)^{(d-1)/d}).$$

We exhaustively guess the coloring of X , which results in at most $3^{|X|} = 2^{\mathcal{O}((\mu \log \mu)^{1-1/d})}$ branches. As X dominates V_3 , after applying the reduction rule R1 to every vertex of X , in each branch there are no vertices with three-element lists. Therefore, the instance obtained in each of the branches is solved in polynomial time using reduction rule R3. The claimed bound follows since $\mu \leq 3n$. ◀

4 Coloring diameter-2 graphs

In this section we prove the main result of the paper, i.e., Theorem 3. Let us recall the following variant of the Chernoff concentration bound.

► **Theorem 8** ([25, Theorem 2.3]). *Let X_1, \dots, X_n be independent random variables with $0 \leq X_i \leq 1$ for each i . Let $X = \sum X_i$ and $\bar{X} = \mathbb{E}[X]$.*

(1) *For any $\varepsilon > 0$,*

$$\Pr(X \geq (1 + \varepsilon)\bar{X}) \leq e^{-\frac{\varepsilon^2 \bar{X}}{2(1 + \varepsilon/3)}}.$$

(2) For any $\varepsilon > 0$,

$$\Pr(X \leq (1 - \varepsilon)\overline{X}) \leq e^{-\frac{\varepsilon^2 \overline{X}}{2}}.$$

It will be more convenient to work with random variables for which we only know bounds on the expected value. For this reason we will use the following corollary of Theorem 8.

► **Corollary 9.** Let X_1, \dots, X_n be independent random variables with $0 \leq X_i \leq 1$ for each i . Let $X = \sum X_i$.

(1) For any $\varepsilon > 0$ and $\hat{X} \geq \mathbb{E}[X]$,

$$\Pr(X \geq (1 + \varepsilon)\hat{X}) \leq e^{-\frac{\varepsilon^2 \hat{X}}{2(1+\varepsilon/3)}}.$$

(2) For any $\varepsilon > 0$ and $\hat{X} \leq \mathbb{E}[X]$,

$$\Pr(X \leq (1 - \varepsilon)\hat{X}) \leq e^{-\frac{\varepsilon^2 \hat{X}}{2}}.$$

Proof. Clearly, if $\hat{X} = \mathbb{E}[X]$, then (1) and (2) follow directly from Theorem 8. So since now assume that this is not the case. In order to prove (1) let us consider a random variable $Y = X + Y_1 + Y_2 + \dots + Y_k$, where $k = \lceil \hat{X} - \mathbb{E}[X] \rceil$ and each Y_i is a constant equal to $\frac{\hat{X} - \mathbb{E}[X]}{k}$. Clearly $\mathbb{E}[Y] = \hat{X}$ and $Y \geq X$, so the statement follows by Theorem 8 (1).

For (2) it is enough to apply Theorem 8 (2) for the random variable $Y = X - \frac{\hat{X}}{\mathbb{E}[X]}$. ◀

We start with a technical lemma that is the crucial ingredient of our algorithm.

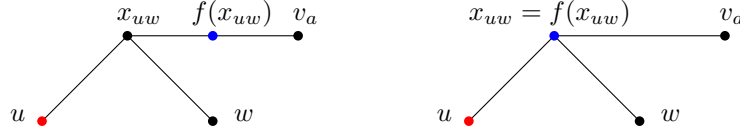
► **Lemma 10.** There exists an absolute constant K such that the following is true. Let G be a 3-colorable graph with n vertices such that

- (i) $\Delta(G) \leq n^{2/3}$,
- (ii) for every $v \in V(G)$, the set $N_G^{\leq 2}(v)$ contains at least $n - \frac{1}{36}n^{2/3}$ vertices,
- (iii) for every two vertices $u, v \in V(G)$ there are at most $n^{2/3}$ vertices w such that $N_G(u) \cap N_G(v) \cap N_G(w) \neq \emptyset$.

Let φ be a proper 3-coloring of G , where $a \in [3]$ is the color that appears most frequently. Define $A := \varphi^{-1}(a)$. Then there exist sets $S \subseteq A$ and $\tilde{S} \subseteq V(G) \setminus A$, each of size at most $K \cdot n^{1/3} \log n$, such that $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$ dominates at least $\frac{n}{6}$ vertices.

Before we prove Lemma 10, let us explain its purpose. Suppose that G is a graph with diameter at most 2 and we are trying to find a 3-coloring of G under the promise that it exists. We start by assigning to each vertex a list of 3 possible colors. Note that if we correctly guess a set S of vertices of the most frequent color a and a set \tilde{S} of vertices together with its coloring using colors $[3] \setminus \{a\}$, then we can deduce the color of each vertex in $N(S) \cap N(\tilde{S})$. Hence, our reduction rules will remove at least one color from the list of each vertex dominated by $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$. If the sets S and \tilde{S} are as in the lemma, then we have just removed at least $\frac{n}{6}$ colors from all the lists by guessing the coloring of only $\mathcal{O}(n^{1/3} \log n)$ vertices. This is roughly why our algorithm is much faster than an exhaustive search.

The assumptions of the lemma can be read as follows: (i) vertices in G do not have too many neighbors, (ii) G is almost a graph with diameter 2 and (iii) common neighbors of every two vertices u and v do not dominate too many vertices of the graph. As we will see later, those assumptions arise naturally when trying to solve the problem using simple branching rules – if any of them is violated, then searching for a 3-coloring of G becomes easier because of other reasons.



■ **Figure 2** The vertex u threatens w : if $f(x_{uw}) \in \tilde{S}$ and $u \in S$, then w has a neighbor with uniquely determined color.

Proof of Lemma 10. Note that we can assume that $n \geq n_0$, where n_0 is a constant that implicitly follows from the reasoning below. Indeed, otherwise it is sufficient to set $K := n_0$, $S := A$, and $\tilde{S} := V(G) \setminus A$. Thus from now on we assume that n is sufficiently large.

For every two vertices $u, v \in V(G)$ such that $N[u] \cap N[v] \neq \emptyset$, let x_{uv} be a vertex from $N[u] \cap N[v]$. Fix some vertex $v_a \in A$ and a function $f : N^{\leq 2}(v_a) \rightarrow N(v_a)$ defined such that $f(u) \in N[u] \cap N(v_a)$.

We start by selecting \tilde{S} as a subset of neighbors of v_a . For such a set \tilde{S} we say that a vertex $u \in A$ *threatens* a vertex $w \in A$ if

- (1) $N[u] \cap N[w] \neq \emptyset$,
- (2) $x_{uw} \in N^{\leq 2}(v_a)$, and
- (3) $f(x_{uw}) \in \tilde{S}$.

Intuitively, u threatens w if selecting u to S would undoubtedly cause w to be dominated by $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$, see Figure 2. The following claim gives us a set \tilde{S} such that each vertex of A is threatened by many vertices.

▷ **Claim 11.** There exists a set $\tilde{S} \subseteq N(v_a)$ of order at most $200n^{1/3} \log n$ such that for at least half of vertices $w \in A$ there are at least $8n^{2/3} \log n$ vertices from A that threaten w .

Proof. We select \tilde{S} randomly in such a way that each neighbor of v_a is included in \tilde{S} independently with probability $\tilde{p} = 100n^{-1/3} \log n$. We will show that \tilde{S} satisfies the desired properties with positive probability.

Note that the size of \tilde{S} is a sum of $\deg(v_a)$ independent random boolean variables and the expected value of $|\tilde{S}|$ is $\deg(v_a) \cdot \tilde{p}$. Recall that by the assumption (i) we have $\deg(v_a) \leq n^{2/3}$. Therefore by Corollary 9 (1) applied with $\varepsilon = 1$ we deduce that

$$\Pr(|\tilde{S}| > 200n^{1/3} \log n) \leq e^{-37.5n^{1/3} \log n}.$$

Let $A' \subseteq A$ be the set of those $v \in A$, for which the set $N(N(v) \setminus N^{\leq 2}(v_a))$ contains fewer than half of vertices from A . We will show that $|A'| \geq \frac{1}{2}|A|$. First, let us estimate the number P of ordered pairs of vertices (u, v) such that u and v have a common neighbor outside of $N^{\leq 2}(v_a)$. By (i) each vertex outside of $N^{\leq 2}(v_a)$ can be a common neighbor for at most $n^{4/3}$ pairs of vertices, so (ii) implies that $P \leq \frac{1}{36}n^2$. Note that a vertex from A is not contained in A' only if it is in at least $|A|$ ordered pairs that contribute to P . It follows that A' contains at least $|A| - \frac{2P}{|A|}$ vertices. Since a is the most frequent color used by the 3-coloring φ , we have $|A| \geq \frac{1}{3}n$, and thus $|A'| \geq \frac{1}{2}|A|$, as desired.

Fix a vertex w from A' . Consider a random variable X_w that counts the number of vertices u from A such that u threatens w and $N(u) \cap N(w) \subseteq N^{\leq 2}(v_a)$. Our plan is to use Corollary 9 to show that X_w is at least $8n^{2/3} \log n$ with high probability.

We start by estimating the expected value of X_w . Let U be the set of vertices $u \in A$ such that $N(u) \cap N(w) \subseteq N^{\leq 2}(v_a)$; note that by the definition of U , there is a vertex in $N[u] \cap N[w] \cap N^{\leq 2}(v_a)$, so x_{uw} and $f(x_{uw})$ exist for all vertices $u \in U$. Each vertex $u \in U$

contributes 1 to X_w if and only if $f(x_{uw}) \in \tilde{S}$, i.e., with probability \tilde{p} . Since $w \in A'$, the size of U is at least $\frac{1}{2}|A|$ minus the number of vertices outside of $N^{\leq 2}(w)$, which totals to at least $\frac{n}{6} - \frac{1}{36}n^{2/3}$ by (ii). Therefore, $\mathbb{E}[X_w] \geq 16n^{2/3} \log n$ for large enough n .

Now we express X_w as a sum of a number of independent random variables. Fix an ordering $t_1, t_2, \dots, t_{\deg(v_a)}$ of neighbors of v_a and define U_i as the set of vertices u from U such that $x_{uw} \in N^{\leq 2}(v_a)$ and $f(x_{uw}) = t_i$. For $i = 1, 2, \dots, \deg(v_a)$ let X_i be a random variable that is equal to $|U_i|$ if $t_i \in \tilde{S}$ and 0 otherwise. Clearly $X_w = \sum_i X_i$ and all the variables $X_1, \dots, X_{\deg(v_a)}$ are independent by the independent selection of \tilde{S} .

By (iii), applied for w and t_i , we obtain that $X_i \leq n^{2/3}$ for all i . Therefore we may use Corollary 9 (2) for the sequence of variables $\frac{X_i}{n^{2/3}}$ and $\varepsilon = \frac{1}{2}$ to deduce that

$$\Pr\left(\frac{X_w}{n^{2/3}} \leq 8 \log n\right) \leq e^{-2 \log n},$$

which gives that

$$\Pr(X_w \leq 8n^{2/3} \log n) \leq n^{-2}.$$

By the union bound we obtain that the probability that \tilde{S} has more than $200n^{1/3} \log n$ vertices or that $X_w < 8n^{2/3} \log n$ for any $w \in A'$ is at most $n \cdot n^{-2} + n^{-37.5n^{1/3}}$. Therefore, for large enough n the set \tilde{S} satisfies the required properties with positive probability, so the proof of the claim is complete. \triangleleft

Having selected \tilde{S} , we proceed to selecting S as a subset of A that guarantees the desired domination property.

\triangleright **Claim 12.** There exists a set $S \subseteq A$ of order at most $2n^{1/3}$ such that at least half of the vertices $w \in A$ are dominated by $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$.

Proof. We randomly select S so that each vertex from A is in S independently with probability $p = n^{-2/3}$. Note that by Corollary 9 (1) the size of S is at most $2n^{1/3}$ with probability at least $1 - e^{-\frac{3}{8}n^{2/3}}$.

Let w be a vertex from A that is threatened by at least $8n^{2/3} \log n$ vertices from A . The probability that w is not dominated by $N(S) \cap N(\tilde{S})$ is at most

$$(1-p)^{8n^{2/3} \log n} \leq e^{-8pn^{2/3} \log n} \leq e^{-8 \log n} \leq n^{-8}.$$

By the union bound it follows that with probability at least $1 - n^{-7}$ all vertices threatened by at least $8n^{2/3} \log n$ vertices from A are dominated by $N(S) \cap N(\tilde{S})$. Claim 11 implies that there are at least $\frac{1}{2}|A|$ such vertices, so the proof is complete. \triangleleft

Setting $K := \max(n_0, 200)$. Now the statement of the lemma follows from Claim 12 by observing that since A is the most frequent color, we have $\frac{1}{2}|A| \geq \frac{1}{6}n$. \blacktriangleleft

Now we are ready to prove Theorem 3.

\blacktriangleright **Theorem 3.** *The LIST 3-COLORING problem on n -vertex graphs with diameter 2 can be solved in time $2^{\mathcal{O}(n^{1/3} \cdot \log^2 n)}$.*

Proof. Let (G, L) be an instance of the LIST 3-COLORING problem. Again, we start by applying reduction rules R1, R2, R3, so we can assume that (G, L) is reduced. Let (V_1, V_2, V_3) be the layer structure of G and set $\mu := |V_3|$.

We use one of the four branching rules to produce a number of instances of the problem, each with fewer vertices with lists of size 3. Those instances are solved recursively and if a success is reported for at least one of them, then the algorithm terminates and reports a success. The following branching rules are applied in the given order – it is essential that B4 is executed only if the rules B1, B2 and B3 cannot be applied.

- B1** If there exists a vertex $v \in V_2 \cup V_3$ such that v has more than $\mu^{2/3}$ neighbors in V_3 , then for every color $a \in L(v)$ solve an instance obtained by replacing $L(v)$ with $\{a\}$ and exhaustively applying the reduction rules.
- B2** If there exists a vertex $v \in V_3$ such that for at least $\frac{1}{36}\mu^{2/3}$ vertices $u \in V_3$ a common neighbor of u and v is in V_2 , then for every color $a \in L(v)$ solve an instance obtained by replacing $L(v)$ with $\{a\}$ and exhaustively applying the reduction rules.
- B3** If there are two vertices $u, v \in V_3$ such that for at least $\mu^{2/3}$ vertices w from V_3 the set $N(u) \cap N(v) \cap N(w)$ is nonempty, then for every two distinct colors a, b construct an instance by setting $L(u) := \{a\}$ and $L(v) := \{b\}$ and one additional instance obtained by replacing vertices u and v with a new vertex z adjacent to $N(u) \cup N(v)$ with $L(z) = [3]$. Apply the reduction rules to each of those instances and solve them recursively.
- B4** Let K be the constant from Lemma 10. For every tuple $(a, S, \tilde{S}, \varphi)$, where
 - $a \in [3]$ is a color,
 - $S \subseteq V_3$ is a set of size at most $K \cdot \mu^{1/3} \log \mu$,
 - $\tilde{S} \subseteq V_3 \setminus S$ is a set of size at most $K \cdot \mu^{1/3} \log \mu$,
 - φ is a coloring of \tilde{S} using colors $[3] \setminus \{a\}$,
 construct an instance by setting $L(v) := \{a\}$ for each $v \in S$ and $L(v) = \{\varphi(v)\}$ for $v \in \tilde{S}$. Apply the reduction rules to each of those instances for which $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$ dominates at least $\frac{1}{6}\mu$ vertices from V_3 and solve them recursively.

Let us show that the above algorithm is correct. Branching rules B1 and B2 are clearly correct, because if there is a solution to the given instance of the LIST 3-COLORING problem, then it assigns to v one color from $L(v)$. The rule B3 is correct because if there is a solution to the given instance of the problem, then it either assigns two different colors to u and v , or assigns the same color to u and v , hence at least one of the constructed instances will admit a solution. Note that contracting the vertices u and v does not increase the diameter. Now consider the branching rule B4. Recall that it is applied only when rules B1, B2 and B3 are inapplicable, so in this case the graph $G[V_3]$ satisfies the assumptions (i)-(iii) of Lemma 10. Therefore if the original instance has a solution, then by Lemma 10 at least one instance constructed in B4 admits a solution. On the other hand, each instance is obtained by fixing the colors of vertices in $S \cup \tilde{S} \subseteq V_3$, so each such a coloring respects lists L . Furthermore, if this coloring is improper, then the application of reductions rules R1 and R2 will cause the algorithm to reject the instance. Hence, the branching rule B4 is correct.

Let us denote by $F(x)$ the maximum running time of the algorithm on an instance with at most x vertices with lists of size 3. By $p(n)$ we denote the cost of exhaustively applying the reduction rules to an instance with n vertices; note that $p(n)$ is polynomial in n .

Now we will bound the running time of the algorithm on our instance (G, L) with μ vertices with lists of size 3, depending on which branching rule was applied.

Case 1: B1 was applied. Note that this branching produced at most three instances of the problem, each with at most $\mu - \mu^{2/3}$ vertices with lists of size 3. This is because for every vertex $u \in V_3$ that is a neighbor of v the color c was removed from $L(u)$. Therefore, in this

case the running time is at most

$$3F\left(\mu - \mu^{2/3}\right) + 3p(n).$$

Case 2: B2 was applied. For a color x let N_x be the number of vertices $u \in V_3$ such that the list of a common neighbor of v and u in V_2 does not contain x . Let us rename the colors by a, b and c such that $N_a \leq N_b \leq N_c$. Note that if a vertex u contributes to N_c , then after the application of reduction rules b (respectively a) is removed from $L(u)$ in the instance constructed for the color a (respectively b). It follows that the running time of the algorithm in this case is at most

$$F(\mu - 1) + 2F\left(\mu - \frac{1}{108}\mu^{2/3}\right) + 3p(n).$$

Case 3: B3 was applied. Let w be a vertex from V_3 such that the set $N(u) \cap N(v) \cap N(w)$ is nonempty. Note that if we set $L(u)$ to $\{a\}$ and $L(v)$ to $\{b\}$, for $a \neq b$, then after applying the reduction rules common neighbors of u and v will have lists of size 1, hence the size of the list of w will be at most 2. Therefore, in this case the running time is at most

$$F(\mu - 1) + 6F\left(\mu - \mu^{2/3}\right) + 7p(n).$$

Case 4: B4 was applied. Note that in the constructed instances, after applying the reduction rules, all vertices from $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$ have lists of size 1, so all vertices dominated by $S \cup \tilde{S} \cup (N(S) \cap N(\tilde{S}))$ have lists of size at most 2. Therefore, all instances that are solved recursively have at most $\mu - \frac{1}{6}\mu$ vertices with lists of size 3. The total number of those instances can be upper bounded by

$$3 \cdot \mu^{2K\mu^{1/3} \log \mu} \cdot 2^{K\mu^{1/3} \log \mu} < 2^{K'\mu^{1/3} \log^2 \mu},$$

for some constant K' . Therefore the total running time in this case is at most

$$2^{K'\mu^{1/3} \log^2 \mu} F\left(\frac{5}{6}\mu\right) + 2^{K'\mu^{1/3} \log^2 \mu} p(n)$$

As the considered cases cover all possibilities, we conclude that $F(\mu)$ is bounded by the maximum of the expressions obtained in all four cases. By solving this recurrence we obtain

$$F(\mu) \leq p(n) \cdot 2^{\mathcal{O}(\mu^{1/3} \log^2 \mu)} = 2^{\mathcal{O}(\mu^{1/3} \log^2 \mu)}.$$

Since $\mu \leq n$, the proof is complete. ◀

5 Possible extensions of our results

We conclude the paper with discussing possible extensions of our results.

5.1 Solving List H -Coloring in small-diameter graphs

For a fixed graph H with possible loops, an instance of LIST H -COLORING is a pair (G, L) , where G is a graph and $L : V(G) \rightarrow 2^{V(H)}$ is a list function. We ask whether there exists a *list homomorphism* from (G, L) to H , i.e., a function $\varphi : V(G) \rightarrow V(H)$, such that (i) for

each $uv \in E(G)$ it holds that $\varphi(u)\varphi(v) \in E(H)$, and (ii) for each $v \in V(G)$ it holds that $\varphi(v) \in L(v)$. Clearly LIST K_k -COLORING is equivalent to LIST k -COLORING. This is why we refer to the vertices of H as *colors*.

We observe that the algorithm from Theorem 2 and Theorem 3 can be adapted to LIST H -COLORING if the graph H satisfies certain conditions. First, the algorithm from Theorem 2 can be adapted to solve the LIST H -COLORING problem if

(P1) every vertex of H has at most two neighbors (possibly including itself, if it is a vertex with a loop).

For such graphs H , once we fix a color of some $v \in V(G)$, all its neighbors have lists of size at most 2.

To adapt the algorithm from Theorem 3, in addition to property (P1), we need two more:

(P2) any two distinct vertices of H must have at most one common neighbor,

(P3) H has no loops.

Property (P2) is needed to ensure that as soon as we fix the coloring of the sets S and \tilde{S} selected in Lemma 10, then the color of every vertex in $N(S) \cap N(\tilde{S})$ is uniquely determined. Property (P3) is needed for our selection of the set \tilde{S} : recall that all these vertices are in the neighborhood of some vertex v_a colored a , which is sufficient to ensure that no vertex of \tilde{S} gets the color a .

Let \mathcal{H} be the family of connected graphs that satisfy property (P1). From the complexity dichotomy for LIST H -COLORING by Feder, Hell, and Huang [13,14] it follows that if $H \in \mathcal{H}$, then LIST H -COLORING is polynomial-time solvable if:

- H has at most two vertices,
- $H = C_4$,
- H is a path,
- H is a path with a loop on one endvertex,

and otherwise the problem is NP-complete and does not admit a subexponential-time algorithm under the ETH. So, in other words, there are two families of graphs $H \in \mathcal{H}$ for which the problem is NP-complete (in general graphs):

- all cycles C_k for $k = 3$ or $k \geq 5$, and
- all graphs obtained from a path with $k \geq 3$ vertices by adding loops on both endvertices; let us call such a graph P_k^* .

Let us present one more simple observation about solving LIST H -COLORING in graphs with small diameter. Consider an instance (G, L) of LIST H -COLORING and suppose that H contains two vertices x, y at distance greater than $\text{diam}(G)$. (Here, with a little abuse of notation, we use the convention that if x and y are in different connected components of H , then their distance is infinite.) We note that there is no (list) homomorphism from G to H that uses both x and y . Thus we can reduce the problem to solving an instance (G, L_x) of LIST $(H - x)$ -COLORING and an instance (G, L_y) of LIST $(H - y)$ -COLORING, where lists L_x (resp. L_y) are obtained from L by removing the vertex x (resp., y) from each set.

Combining all observations above, we obtain the following results. We skip the formal proofs, as they are essentially the same as the ones of Theorem 2 and Theorem 3 and bring no new insight.

► **Theorem 13.** *Let $H \in \mathcal{H}$. Consider an instance (G, L) of LIST H -COLORING, where G is of diameter 2. Then (G, L) can be solved*

1. *in polynomial time if $H \notin \{C_3, C_5, P_3^*\}$,*
2. *in time $2^{\mathcal{O}(n^{1/3} \log^2 n)}$ if $H \in \{C_3, C_5\}$,*
3. *in time $2^{\mathcal{O}(n^{1/2} \log^{1/2} n)}$ if $H = P_3^*$.*

► **Theorem 14.** Let $H \in \mathcal{H}$. Consider an instance (G, L) of LIST H -COLORING, where G is of diameter 3. Then (G, L) can be solved

1. in polynomial time if $H \notin \{C_3, C_5, C_6, C_7, P_3^*, P_4^*\}$,
2. in time $2^{\mathcal{O}(n^{2/3} \log^{2/3} n)}$ if $H \in \{C_3, C_5, C_6, C_7, P_3^*, P_4^*\}$.

5.2 Weighted coloring problems

Another possible generalization of LIST 3-COLORING would be to introduce weights: for each pair (v, c) , where $v \in V(G)$ and $c \in \{1, 2, 3\}$, we are given a cost $\mathbf{w}(v, c)$ of coloring v with c , and we ask for a proper coloring minimizing the total cost. A natural special case of this problem is INDEPENDENT ODD CYCLE TRANSVERSAL, where we ask for a minimum-sized independent set which intersects all odd cycles.

Let us point out that the branching phases in our algorithms from Theorem 2 and Theorem 3 can handle this type of modification. However, this is no longer the case for the last phase, when the problem of coloring a graph with all lists of size at most two is reduced to 2-SAT using Theorem 4. It is known that a weighted variant of 2-SAT is NP-complete and admits no subexponential-time algorithm, unless the ETH fails [28]. Thus, in order to extend our algorithmic results to weighted setting, we need to find a way to replace using Theorem 4 with some other strategy of dealing with lists of size 2.

References

- 1 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.
- 2 Marthe Bonamy, Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Panos Giannopoulos, Eun Jung Kim, Paweł Rzażewski, Florian Sikora, and Stéphan Thomassé. EPTAS and subexponential algorithm for Maximum Clique on disk and unit ball graphs. *J. ACM*, 68(2):9:1–9:38, 2021. doi:10.1145/3433160.
- 3 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex sets for graphs of bounded diameter. *Inf. Process. Lett.*, 131:26–32, 2018. doi:10.1016/j.ipl.2017.11.004.
- 4 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for P_5 -free graphs. *Algorithmica*, 81(4):1342–1369, 2019. doi:10.1007/s00453-018-0474-x.
- 5 Christoph Brause, Petr Golovach, Barnaby Martin, Daniël Paulusma, and Siani Smith. Acyclic, star, and injective colouring: Bounding the diameter. *CoRR*, abs/2104.10593, 2021. arXiv:2104.10593.
- 6 Victor A. Campos, Guilherme de C. M. Gomes, Allen Ibiapina, Raul Lopes, Ignasi Sau, and Ana Silva. Coloring problems on bipartite graphs of small diameter. *CoRR*, abs/2004.11173, 2020. arXiv:2004.11173.
- 7 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. Math. (2)*, 164(1):51–229, 2006.
- 8 Maria Chudnovsky and Paul D. Seymour. The three-in-a-tree problem. *Comb.*, 30(4):387–417, 2010. doi:10.1007/s00493-010-2334-4.
- 9 Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Clique-width for hereditary graph classes. *CoRR*, abs/1901.00335, 2019. arXiv:1901.00335.
- 10 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for exponential-time-hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49(6):1291–1331, 2020. doi:10.1137/20M1320870.

- 11 Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 637–646. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.14.
- 12 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.
- 13 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Comb.*, 19(4):487–505, 1999. doi:10.1007/s004939970003.
- 14 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.
- 15 Fedor V. Fomin, Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensionality. In *Encyclopedia of Algorithms*, pages 203–207. Springer, 2016. doi:10.1007/978-1-4939-2864-4_47.
- 16 Peter Gartland and Daniel Lokshantov. Independent set on P_k -free graphs in quasi-polynomial time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.
- 17 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *J. Graph Theory*, 84(4):331–363, 2017. doi:10.1002/jgt.22028.
- 18 M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 325–356. North-Holland, 1984. doi:10.1016/S0304-0208(08)72943-8.
- 19 Jan Kratochvíl. Can they cross? and how?: (the hitchhiker’s guide to the universe of geometric intersection graphs). In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 75–76. ACM, 2011. doi:10.1145/1998196.1998208.
- 20 Jan Kratochvíl and Jirí Matousek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994. doi:10.1006/jctb.1994.1071.
- 21 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring H -free graphs of bounded diameter. In Peter Rossmanith, Pinar Heggenes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.14.
- 22 Barnaby Martin, Daniël Paulusma, and Siani Smith. Computing independent transversals for H -free graphs of bounded diameter. Abstract at Workshop on Graph Modification: algorithms, experiments and new problems, 23rd - 24th January 2020, Bergen, Norway, 2020. URL: <https://graphmodif.iu.uib.no/abs/fri-2.pdf>.
- 23 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring graphs of bounded diameter in the absence of small cycles. *CoRR*, abs/2101.07856, 2021. arXiv:2101.07856.
- 24 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 865–877. Springer, 2015. doi:10.1007/978-3-662-48350-3_72.
- 25 Colin McDiarmid. Concentration. In J. Ramirez-Alfonsin M. Habib, C. McDiarmid and B. Reed, editors, *Probabilistic methods for algorithmic discrete mathematics*, volume 16 of *Algorithms and Combinatorics*, pages 195–248. Springer, 1998.
- 26 George B. Mertzios and Paul G. Spirakis. Algorithms and almost tight results for 3-colorability of small diameter graphs. *Algorithmica*, 74(1):385–414, 2016. doi:10.1007/s00453-014-9949-6.

- 27 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 28 Stefan Porschen. On variable-weighted exact satisfiability problems. *Ann. Math. Artif. Intell.*, 51(1):27–54, 2007. doi:10.1007/s10472-007-9084-z.
- 29 Neil Robertson and Paul D. Seymour. Graph minors. v. excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.
- 30 Sebastian Schnettler. A structured overview of 50 years of small-world research. *Soc. Networks*, 31(3):165–178, 2009. doi:10.1016/j.socnet.2008.12.004.

Stability Yields Sublinear Time Algorithms for Geometric Optimization in Machine Learning

Hu Ding 

School of Computer Science and Technology,
University of Science and Technology of China, Anhui, China

Abstract

In this paper, we study several important geometric optimization problems arising in machine learning. First, we revisit the Minimum Enclosing Ball (MEB) problem in Euclidean space \mathbb{R}^d . The problem has been extensively studied before, but real-world machine learning tasks often need to handle large-scale datasets so that we cannot even afford linear time algorithms. Motivated by the recent developments on beyond worst-case analysis, we introduce the notion of stability for MEB, which is natural and easy to understand. Roughly speaking, an instance of MEB is stable, if the radius of the resulting ball cannot be significantly reduced by removing a small fraction of the input points. Under the stability assumption, we present two sampling algorithms for computing radius-approximate MEB with sample complexities independent of the number of input points n . In particular, the second algorithm has the sample complexity even independent of the dimensionality d . We also consider the general case without the stability assumption. We present a hybrid algorithm that can output either a radius-approximate MEB or a covering-approximate MEB, which improves the running time and the number of passes for the previous sublinear MEB algorithms. Further, we extend our proposed notion of stability and design sublinear time algorithms for other geometric optimization problems including MEB with outliers, polytope distance, one-class and two-class linear SVMs (without or with outliers). Our proposed algorithms also work fine for kernels.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases stability, sublinear time, geometric optimization, machine learning

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.38

Acknowledgements The author wants to thank Jinhui Xu and the anonymous reviewers for their helpful comments and suggestions for improving the paper.

1 Introduction

Many real-world machine learning tasks can be formulated as geometric optimization problems in Euclidean space. We start with a fundamental geometric optimization problem, *Minimum Enclosing Ball (MEB)*, which has attracted a lot of attentions in past years. Given a set P of n points in Euclidean space \mathbb{R}^d , where d could be quite high, the problem of MEB is to find a ball with minimum radius to cover all the points in P [16, 38, 60]. MEB finds several important applications in machine learning [68]. For example, the popular classification model *Support Vector Machine (SVM)* can be formulated as an MEB problem in high dimensional space, if all the mapped points have the same norm by using kernel method, *e.g.*, the popular radial basis function kernel [80]. Hence fast MEB algorithms can be adopted to speed up its training procedure [24, 25, 80]. Recently, MEB has also been used for preserving privacy [37, 69] and quantum cryptography [46].

Usually, we consider the approximate solutions of MEB. If a ball covers all the n points but has a radius larger than the optimal one, we call it a “**radius-approximate solution**”; if a ball has the radius no larger than the optimal one but covers less than n points, we call it a “**covering-approximate solution**” instead (the formal definitions are shown in Section 3). In the era of big data, the dataset could be so large that we cannot even afford linear time algorithms. This motivates us to ask the following questions:



© Hu Ding;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 38; pp. 38:1–38:19

Leibniz International Proceedings in Informatics



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

Is it possible to develop approximation algorithms for MEB that run in sublinear time in the input size? And how about other high-dimensional geometric optimization problems arising in machine learning?

It is common to assume that the input data is represented by a $n \times d$ matrix, and thus any algorithm having complexity $o(nd)$ can be considered as a sublinear time algorithm. In practice, data items are usually represented as sparse vectors in \mathbb{R}^d ; thus it can be fast to perform the operations, like distance computing, even though the dimensionality d is high (see the concluding remarks of [25]). Moreover, the number of input points n is often much larger than the dimensionality d in many real-world scenarios. **Therefore, we are interested in designing the algorithms that have complexities sublinear in n (or linear in n but with small factor before it).** Designing sublinear time algorithms has become a promising approach to handle many big data problems, and a detailed discussion on previous works is given in Section 2.

1.1 Our Main Ideas and Results

Our idea for designing sublinear time MEB algorithms is inspired by the recent developments on optimization with respect to stable instances, under the umbrella of *beyond worst-case analysis* [74]. For example, several recent works introduced the notion of stability for problems like clustering and max-cut [8, 10, 15]. In this paper, we give the notion of “stability” for MEB. Roughly speaking, an instance of MEB is stable, if the radius of the resulting ball cannot be significantly reduced by removing a small fraction of the input points (*e.g.*, the radius cannot be reduced by 10% if only 1% of the points are removed). The rationale behind this notion is quite natural: if the given instance is not stable, the small fraction of points causing significant reduction in the radius should be viewed as outliers (or we may need multiple balls to cover the input points as k -center clustering [45, 52]). To the best of our knowledge, this is the first study on MEB from the perspective of stability.

We prove an important implication of the stability assumption: informally speaking, if an instance of MEB is stable, its center should reveal a certain extent of robustness in the space (Section 4). Using this implication, we propose two sampling algorithms for computing $(1 + \epsilon)$ -radius approximate MEB with sublinear time complexities (Section 5); in particular, our second algorithm has the sample size (*i.e.*, the number of sampled points) independent of the number of input points n and dimensionality d (to the best of our knowledge, this is the first algorithm achieving $(1 + \epsilon)$ -radius approximation with such a sublinear complexity).

Moreover, we have an interesting observation: the ideas developed under the stability assumption can even help us to solve the general instance without the stability assumption, if we relax the requirement slightly. We introduce a hybrid approach that can output either a radius-approximate MEB or a covering-approximate MEB, depending upon whether the input instance is sufficiently stable¹ (Section 6). It is worth noting that the simple uniform sampling idea based on VC-dimension [49, 81] can only yield a “bi-criteria” approximation, which has errors on both the radius and the number of covered points (see the discussion on our first sampling algorithm in Section 5.1). Comparing with the sublinear time MEB algorithm proposed by Clarkson *et al.* [25], we reduce the total running time from $\tilde{O}(\epsilon^{-2}n + \epsilon^{-1}d + M)$ to $O(n + h(\epsilon, \delta) \cdot d + M)$, where M is the number of non-zero entries in the input $n \times d$ matrix and $h(\epsilon, \delta)$ is a factor depending on the pre-specified radius error bound ϵ and covering error bound δ . Thus, our improvement is significant if $n \gg d$. The only tradeoff is that we allow a

¹ We do not need to know whether the instance is stable or not, when running our algorithm.

covering approximation for unstable instance (given the lower bound proved by [25], it is quite unlikely to reduce the term $\epsilon^{-2}n$ if we keep restricting the output to be $(1 + \epsilon)$ -radius approximation). Moreover, our algorithm only needs uniform sampling and a single pass over the data; on the other hand, the algorithm of [25] needs $\tilde{O}(\epsilon^{-1})$ passes (the details are shown in Table 1).

■ **Table 1** The existing and our results for computing MEB in high dimensions. In the table, “rad.” and “cov.” stand for “radius approximation” and “covering approximation”, respectively. “ M ” is the number of non-zero entries in the input $n \times d$ matrix. The factor C_1 depends on ϵ and the stability degree of the given instance; the factor C_2 depends on ϵ and δ . The mark “*” means that the method can be extended for MEB with outliers.

Results		Quality	Time	Number of passes
Clarkson <i>et al.</i> [25]		$(1 + \epsilon)$ -rad.	$\tilde{O}(\epsilon^{-2}n + \epsilon^{-1}d + M)$	$\tilde{O}(\epsilon^{-1})$
Core-sets methods* [16, 24, 60, 71]		$(1 + \epsilon)$ -rad.	roughly $O(\epsilon^{-1}nd)$ or $O(\epsilon^{-1}(n + d + M))$ if $M = o(nd)$	$O(\epsilon^{-1})$
Numerical method [76]		$(1 + \epsilon)$ -rad.	$\tilde{O}(\epsilon^{-1/2}nd)$ or $\tilde{O}(\epsilon^{-1/2}(n + d + M))$ if $M = o(nd)$	$O(\epsilon^{-1/2})$
Numerical method [6]		$(1 + \epsilon)$ -rad.	$\tilde{O}(nd + n\sqrt{d}/\sqrt{\epsilon})$	$\tilde{O}(d + \sqrt{d/\epsilon})$
Streaming algorithm [4, 21]		1.22-rad.	$O(nd/\epsilon^5)$	one pass
This paper	stable instance*	$(1 + \epsilon)$ -rad.	$O(C_1 \cdot d)$	uniform sampling
	general instance*	$(1 + \epsilon)$ -rad. or $(1 - \delta)$ -cov.	$O((n + C_2)d)$ or $O(n + C_2 \cdot d + M)$ if $M = o(nd)$	uniform sampling plus a single pass

Our proposed notion of stability can be naturally extended to several other geometric optimization problems arising in machine learning.

MEB with outliers. In practice, we often assume the presence of outliers in given datasets. MEB with outliers is a natural generalization of the MEB problem, where the goal is to find the minimum ball covering at least a certain fraction of input points. The presence of outliers makes the problem not only non-convex but also highly combinatorial in high dimensions. We define the stability for MEB with outliers, and propose the sublinear time approximation algorithms. Our algorithms are the first sublinear time **single-criterion** approximation algorithms for the MEB with outliers problem (comparing with the previous bi-criteria approximations like [18, 31]), to the best of our knowledge.

Polytope distance and SVM. Given a set P of points in \mathbb{R}^d , the *polytope distance* problem is to compute the shortest distance of any point inside the convex hull of P to the origin. Similar to MEB, polytope distance is also a fundamental problem in computational geometry and has many important applications, such as *sparse approximation* [24]. The polytope distance problem is also closely related to SVMs. Actually, training linear SVM is equivalent to solving the polytope distance problem for the Minkowski difference of two differently

labeled training datasets [41]. Though polytope distance is quite different from the MEB problem, they in fact share several common features. For instance, both of them can be solved by the greedy core-set construction method [24]. Following our ideas for MEB, we define the stability for polytope distance, and propose the sublinear time algorithms.

Because the geometric optimization problems studied in this paper are motivated from machine learning applications, we also take into account the **kernels** [78]. Our proposed algorithms only need to conduct the basic operations, like computing the distance and inner product, on the data items. Therefore, they also work fine for kernels.

The rest of the paper is organized as follows. In Section 2, we summarize the previous results that are related to our work. In Section 3, we present the important definitions and briefly introduce the coresets construction method for MEB from [16] (which will be used in our following algorithms design and analysis). In Section 4, we prove the implication of MEB stability. Further, we propose two sublinear time MEB algorithms in Section 5. We also briefly introduce several important extensions in Section 6; due to the space limit, we leave the details to our full paper.

2 Previous Work

The works most related to ours are [7, 25]. Clarkson *et al.* [25] developed an elegant perceptron framework for solving several optimization problems arising in machine learning, such as MEB. Given a set of n points in \mathbb{R}^d represented as an $n \times d$ matrix with M non-zero entries, their framework can compute the MEB in $\tilde{O}(\frac{n}{\epsilon^2} + \frac{d}{\epsilon})$ time². Note that the parameter “ ϵ ” is an additive error (*i.e.*, the resulting radius is $r + \epsilon$ if r is the radius of the optimal MEB) which can be converted into a relative error (*i.e.*, $(1 + \epsilon)r$) in $O(M)$ preprocessing time. Thus, if $M = o(nd)$, the running time is still sublinear in the input size nd (please see Table 1). The framework of [25] also inspires the sublinear time algorithms for training SVMs [51] and approximating Semidefinite Programs [40]. Hayashi and Yoshida [50] presented a sampling-based method for minimizing quadratic functions of which the MEB objective is a special case, but it yields a large additive error $O(\epsilon n^2)$.

Alon *et al.* [7] studied the following property testing problem: given a set of n points in some metric space, determine whether the instance is (k, b) -clusterable, where an instance is called (k, b) -clusterable if it can be covered by k balls with radius (or diameter) $b > 0$. They proposed several sampling algorithms to answer the question “approximately”. Specifically, they distinguish between the case that the instance is (k, b) -clusterable and the case that it is ϵ -far away from (k, b') -clusterable, where $\epsilon \in (0, 1)$ and $b' \geq b$. “ ϵ -far” means that more than ϵn points should be removed so that it becomes (k, b') -clusterable. Note that their method cannot yield a single criterion radius-approximation or covering-approximation algorithm for the MEB problem, since it will introduce unavoidable errors on the radius and the number of covered points due to the relaxation of “ ϵ -far”. But it is possible to convert it into a “**bi-criteria**” approximation, where it allows approximations on both the radius and the number of uncovered outliers (*e.g.*, discard more than the pre-specified number of outliers).

MEB and core-set. A *core-set* is a small set of points that approximates the structure/shape of a much larger point set [1, 35, 72]. The core-set idea has also been used to compute approximate MEB in high dimensional space [18, 57, 60, 71]. Bădoiu and Clarkson [16] showed

² The asymptotic notation $\tilde{O}(f) = O(f \cdot \text{polylog}(\frac{nd}{\epsilon}))$.

that it is possible to find a core-set of size $\lceil 2/\epsilon \rceil$ that yields a $(1 + \epsilon)$ -radius approximate MEB. Several other methods can yield even lower core-set sizes, such as [17, 57]. In fact, the algorithm for computing the core-set of MEB is a *Frank-Wolfe* algorithm [39], which has been systematically studied by Clarkson [24]. Other MEB algorithms that do not rely on core-sets include [6, 38, 76]. Agarwal and Sharathkumar [4] presented a streaming $(\frac{1+\sqrt{3}}{2} + \epsilon)$ -radius approximation algorithm for computing MEB; later, Chan and Pathak [21] proved that the same algorithm actually yields an approximation ratio less than 1.22.

MEB with outliers and bi-criteria approximations. The MEB with outliers problem can be viewed as the case $k = 1$ of the k -center clustering with outliers problem [22]. Bădoiu *et al.* [18] extended their core-set idea to the problems of MEB and k -center clustering with outliers, and achieved linear time bi-criteria approximation algorithms (if k is assumed to be a constant). Huang *et al.* [53] and Ding *et al.* [31, 33] respectively showed that simple uniform sampling approach can yield bi-criteria approximation of k -center clustering with outliers. Several algorithms for the low dimensional MEB with outliers have also been developed [5, 34, 47, 62]. There also exist a number of works on streaming MEB and k -center clustering with outliers [20, 23, 63, 82]. Other related topics include robust optimization [14], robust fitting [3, 48], and optimization with uncertainty [19].

Polytope distance and SVMs. The Gilbert's algorithm [42] is one of the earliest known algorithms for computing polytope distance. Similar to the core-set construction of MEB, the Gilbert's algorithm is also an instance of the Frank-Wolfe algorithm where the upper bound of the number of iterations is independent of the data size and dimensionality [24, 41]. In general, SVM can be formulated as a quadratic programming problem, and a number of efficient techniques have been developed besides the Gilbert's algorithm, such as the soft margin SVM [26, 73], ν -SVM [27, 77], and CVM [80].

Optimizations under stability. Bilu and Linial [15] showed that the Max-Cut problem becomes easier if the given instance is stable with respect to perturbation on edge weights. Ostrovsky *et al.* [70] proposed a separation condition for k -means clustering which refers to the scenario where the clustering cost of k -means is significantly lower than that of $(k - 1)$ -means for a given instance, and demonstrated the effectiveness of the Lloyd heuristic [61] under the separation condition. Balcan *et al.* [10] introduced the concept of approximation-stability for finding the ground-truth of k -median and k -means clustering. Awasthi *et al.* [8] introduced another notion of clustering stability and gave a PTAS for k -median and k -means clustering. More clustering algorithms under stability assumption were studied in [9, 11–13, 59].

Sublinear time algorithms. Besides the aforementioned sublinear MEB algorithm [25], a number of sublinear time algorithms have been studied for the problems like clustering [29, 54, 55, 64, 65] and property testing [44]. More detailed discussion on sublinear time algorithms can be found in the survey papers [28, 75].

3 Definitions and Preliminaries

We describe and analyze our algorithms in the unit-cost RAM model [66]. Suppose the input is represented by an $n \times d$ matrix (*i.e.*, n points in \mathbb{R}^d). As mentioned in [25], it is common to assume that any entry of the matrix can be recovered in constant time.

We let $|A|$ denote the number of points of a given point set A in \mathbb{R}^d , and $\|x - y\|$ denote the Euclidean distance between two points x and y in \mathbb{R}^d . We use $\mathbb{B}(c, r)$ to denote the ball centered at a point c with radius $r > 0$. Below, we give the definitions for MEB and the notion of stability. To keep the structure of our paper more compact, we place other necessary definitions for our extensions to the full paper.

► **Definition 1** (Minimum Enclosing Ball (MEB)). *Given a set P of n points in \mathbb{R}^d , the MEB problem is to find a ball with minimum radius to cover all the points in P . The resulting ball and its radius are denoted by $\mathbf{MEB}(P)$ and $\mathbf{Rad}(P)$, respectively.*

► **Definition 2** (Radius Approximation and Covering Approximation). *Let $0 < \epsilon, \delta < 1$. A ball $\mathbb{B}(c, r)$ is called a $(1 + \epsilon)$ -radius approximation of $\mathbf{MEB}(P)$, if the ball covers all points in P and has radius $r \leq (1 + \epsilon)\mathbf{Rad}(P)$. On the other hand, the ball is called a $(1 - \delta)$ -covering approximation of $\mathbf{MEB}(P)$, if it covers at least $(1 - \delta)n$ points in P and has radius $r \leq \mathbf{Rad}(P)$.*

Both radius approximation and covering approximation are single-criterion approximations. When ϵ (resp., δ) approaches to 0, the $(1 + \epsilon)$ -radius approximation (resp., $(1 - \delta)$ -covering approximation) will approach to $\mathbf{MEB}(P)$. The “covering approximation” seems to be similar to “MEB with outliers”, but actually they are quite different.

► **Definition 3** ((α, β) -stable). *Given a set P of n points in \mathbb{R}^d with two parameters α and β in $(0, 1)$, P is an (α, β) -stable instance if (1) $\mathbf{Rad}(P') > (1 - \alpha)\mathbf{Rad}(P)$ for any $P' \subset P$ with $|P'| > (1 - \beta)n$, and (2) there exists a $P'' \subset P$ with $|P''| = (1 - \beta)n$ having $\mathbf{Rad}(P'') \leq (1 - \alpha)\mathbf{Rad}(P)$.*

The intuition of Definition 3. Actually, β can be viewed as a function of α . For any $\alpha > 0$, there always exists a $\beta \geq \frac{1}{n}$ such that P is an (α, β) -stable instance ($\beta \geq \frac{1}{n}$ because we must remove at least one point). The property of stability indicates that $\mathbf{Rad}(P)$ cannot be significantly reduced unless removing a large enough fraction of points from P . For a fixed α , the larger β is, the more stable P becomes. Actually, our stability assumption is quite reasonable in practice. For example, if the radius can be reduced considerably (say by $\alpha = 10\%$) after removing only a very small fraction (say $\beta = 1\%$) of points, it is natural to view the small fraction of points as outliers. In practice, it is difficult to obtain the exact value of β for a fixed α . However, the value of β only affects the sample sizes in our proposed algorithms in Section 5, and thus only assuming a reasonable lower bound $\beta_0 < \beta$ is already sufficient. To better understand the notion of stability in high dimensions, we consider the following two examples.

Example (i). Suppose that the distribution of P is uniform and dense inside $\mathbf{MEB}(P)$. Let $\alpha \in (0, 1)$ be a fixed number, and we study the corresponding β of P . If we want the radius of the remaining $(1 - \beta)n$ points to be as small as possible, intuitively we should remove the outermost βn points (since P is uniform and dense). Let P'' denote the set of innermost $(1 - \beta)n$ points that has $\mathbf{Rad}(P'') \leq (1 - \alpha)\mathbf{Rad}(P)$. Then we have $\frac{|P''|}{|P|} \approx \frac{\text{Vol}(\mathbf{MEB}(P''))}{\text{Vol}(\mathbf{MEB}(P))} = \frac{(\mathbf{Rad}(P''))^d}{(\mathbf{Rad}(P))^d} \leq (1 - \alpha)^d$, where $\text{Vol}(\cdot)$ is the volume function. That is, $1 - \beta \leq (1 - \alpha)^d$ and thus $\lim_{d \rightarrow \infty} \beta = 1$ when α is fixed; that means P tends to be very stable as d increases.

Example (ii). Consider a regular d -dimensional simplex P containing $d + 1$ points where each pair of points have the pairwise distance equal to 1. It is not hard to obtain $\mathbf{Rad}(P) = \sqrt{\frac{d}{2(1+d)}}$, and we denote it by r_d . If we remove $\beta(d + 1)$ points from P , namely it becomes a regular d' -dimensional simplex with $d' = (1 - \beta)(d + 1) - 1$, the new radius $r_{d'} = \sqrt{\frac{d'}{2(1+d')}}$. To achieve $\frac{r_{d'}}{r_d} \leq 1 - \alpha$ with a fixed α , it is easy to see that $1 - \beta$ should be no larger than $\frac{1}{1+(2\alpha-\alpha^2)d}$ and thus $\lim_{d \rightarrow \infty} \beta = 1$. Similar to example (i), the instance P tends to be very stable as d increases.

3.1 Core-set Construction for MEB [16]

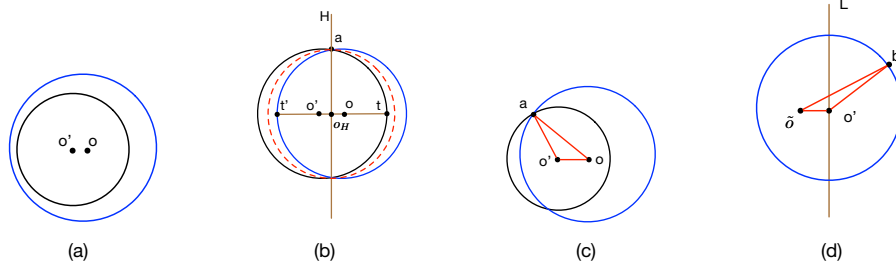
To compute a $(1 + \epsilon)$ -radius approximate MEB, Bădoiu and Clarkson [16] proposed an algorithm yielding an MEB core-set of size $2/\epsilon$ (for convenience, we always assume that $2/\epsilon$ is an integer). We first briefly introduce their main idea, since it will be used in our proposed algorithms (we do not use the MEB algorithm of [24] because it is not quite convenient to analyze under our stability assumption; the other construction algorithms [17, 57], though achieving lower core-set sizes, are more complicated and thus not applicable to our problems).

Given a point set $P \subset \mathbb{R}^d$, the algorithm is a simple iterative procedure. Initially, it selects an arbitrary point from P and places it into an initially empty set T . In each of the following $2/\epsilon$ iterations, the algorithm updates the center of $\mathbf{MEB}(T)$ and adds to T the farthest point from the current center of $\mathbf{MEB}(T)$. Finally, the center of $\mathbf{MEB}(T)$ induces a $(1 + \epsilon)$ -radius approximation for $\mathbf{MEB}(P)$. The selected set of $2/\epsilon$ points (*i.e.*, T) is called the core-set of MEB. However, computing the exact center of $\mathbf{MEB}(T)$ could be expensive; in practice, one may only compute an approximate center of $\mathbf{MEB}(T)$ in each iteration. In the i -th iteration, we let c_i denote the exact center of $\mathbf{MEB}(T)$; also, let r_i be the radius of $\mathbf{MEB}(T)$. Suppose ξ is a given number in $(0, 1)$. Using another algorithm proposed in [16, Section 3], one can compute an approximate center o_i having the distance to c_i less than ξr_i in $O(\frac{1}{\xi^2}|T|d)$ time. Since we only compute o_i rather than c_i in each iteration, we in fact only select the farthest point to o_i (not c_i). In [31], Ding provided a more careful analysis on Bădoiu and Clarkson's method and presented the following theorem.

► **Theorem 4** ([31]). *In the core-set construction algorithm of [16], if one computes an approximate MEB for T in each iteration and the resulting center o_i has the distance to c_i less than ξr_i with $\xi = s \frac{\epsilon}{1+\epsilon}$ for some $s \in (0, 1)$, the final core-set size is bounded by $z = \frac{2}{(1-s)\epsilon}$. Also, the bound could be arbitrarily close to $2/\epsilon$ when s is sufficiently small.*

► **Remark 5.** (i) We can simply set s to be any constant in $(0, 1)$; for instance, if $s = 1/3$, the core-set size will be bounded by $z = 3/\epsilon$. Since $|T| \leq z$ in each iteration, the total running time is $O\left(z(|P|d + \frac{1}{\xi^2}zd)\right) = O\left(\frac{1}{\epsilon}(|P| + \frac{1}{\epsilon^3})d\right)$. (ii) We also want to emphasize a simple observation mentioned in [18, 31] on the above core-set construction procedure, which will be used in our algorithms and analyses later on. The algorithm always selects the farthest point to o_i in each iteration. However, this is actually not necessary. As long as the selected point has distance at least $(1 + \epsilon)\mathbf{Rad}(P)$, the result presented in Theorem 4 is still true. If no such a point exists (*i.e.*, $P \setminus \mathbb{B}(o_i, (1 + \epsilon)\mathbf{Rad}(P)) = \emptyset$), a $(1 + \epsilon)$ -radius approximate MEB (*i.e.*, the ball $\mathbb{B}(o_i, (1 + \epsilon)\mathbf{Rad}(P))$) has been already obtained.

► **Remark 6 (kernels).** If each point $p \in P$ is mapped to $\psi(p)$ in \mathbb{R}^D by some kernel function (*e.g.*, as the CVM [80]), where D could be $+\infty$, we can still run the core-set algorithm of [16, 58], since the algorithm only needs to compute the distances and the center o_i is always a convex combination of T in each iteration; instead of returning an explicit center, the algorithm will output the coefficients of the convex combination for the center. And similarly, our Algorithm 2 presented in Section 5.2 also works fine for kernels.



■ **Figure 1** (a) The case $\text{MEB}(P') \subset \text{MEB}(P)$; (b) an illustration under the assumption $\angle ao'o < \pi/2$ in the proof of Claim 9; (c) the angle $\angle ao'o \geq \pi/2$; (d) an illustration of Lemma 10.

4 Implication of the Stability Property

In this section, we show an important implication of the stability property of Definition 3.

► **Theorem 7.** Assume $\epsilon, \epsilon', \beta_0 \in (0, 1)$. Let P be an (ϵ^2, β) -stable instance of the MEB problem with $\beta > \beta_0$, and o be the center of its MEB. Let \tilde{o} be a given point in \mathbb{R}^d . Assume the number $r \leq (1 + \epsilon'^2)\text{Rad}(P)$. If the ball $\mathbb{B}(\tilde{o}, r)$ covers at least $(1 - \beta_0)n$ points from P , the following holds

$$\|\tilde{o} - o\| < (2\sqrt{2}\epsilon + \sqrt{3}\epsilon')\text{Rad}(P). \quad (1)$$

Theorem 7 indicates that if a ball covers a large enough subset of P and its radius is bounded, its center should be close to the center of $\text{MEB}(P)$. Let $P' = \mathbb{B}(\tilde{o}, r) \cap P$, and assume o' is the center of $\text{MEB}(P')$. To bound the distance between \tilde{o} and o , we bridge them by the point o' (since $\|\tilde{o} - o\| \leq \|\tilde{o} - o'\| + \|o' - o\|$). The following are two key lemmas for proving Theorem 7.

► **Lemma 8.** The distance $\|o' - o\| \leq \sqrt{2}\epsilon\text{Rad}(P)$.

Proof. We consider two cases: $\text{MEB}(P')$ is totally covered by $\text{MEB}(P)$ and otherwise. For the first case (see Figure 1(a)), it is easy to see that

$$\|o' - o\| \leq \text{Rad}(P) - (1 - \epsilon^2)\text{Rad}(P) = \epsilon^2\text{Rad}(P) < \sqrt{2}\epsilon\text{Rad}(P), \quad (2)$$

where the first inequality comes from the fact that $\text{MEB}(P')$ has radius at least $(1 - \epsilon^2)\text{Rad}(P)$ (Definition 3). Thus, we can focus on the second case below.

Let a be any point located on the intersection of the two spheres of $\text{MEB}(P')$ and $\text{MEB}(P)$. Consequently, we have the following claim.

▷ **Claim 9.** The angle $\angle ao'o \geq \pi/2$.

Proof. Suppose that $\angle ao'o < \pi/2$. Note that $\angle ao'o$ is always smaller than $\pi/2$ since $\|o - a\| = \text{Rad}(P) \geq \text{Rad}(P') = \|o' - a\|$. Therefore, o and o' are separated by the hyperplane H that is orthogonal to the segment $\overline{o'o}$ and passes through the point a . See Figure 1(b). Now we show that P' can be covered by a ball smaller than $\text{MEB}(P')$. Let o_H be the point $H \cap \overline{o'o}$, and t (resp., t') be the point collinear with o and o' on the right side of the sphere of $\text{MEB}(P')$ (resp., left side of the sphere of $\text{MEB}(P)$; see Figure 1(b)). Then, we have

$$\begin{aligned} \|t - o_H\| + \|o_H - o'\| &= \|t - o'\| = \|a - o'\| < \|o' - o_H\| + \|o_H - a\| \\ \implies \|t - o_H\| &< \|o_H - a\|. \end{aligned} \quad (3)$$

Similarly, we have $\|t' - o_H\| < \|o_H - a\|$. Consequently, $\mathbf{MEB}(P) \cap \mathbf{MEB}(P')$ is covered by the ball $\mathbb{B}(o_H, \|o_H - a\|)$. Further, because P' is covered by $\mathbf{MEB}(P) \cap \mathbf{MEB}(P')$ and $\|o_H - a\| < \|o' - a\| = \mathbf{Rad}(P')$, P' is covered by the ball $\mathbb{B}(o_H, \|o_H - a\|)$ that is smaller than $\mathbf{MEB}(P')$. This contradicts to the fact that $\mathbf{MEB}(P')$ is the minimum enclosing ball of P' . Thus, the claim $\angle ao'o \geq \pi/2$ is true. \triangleleft

Given Claim 9, we know that $\|o' - o\| \leq \sqrt{(\mathbf{Rad}(P))^2 - (\mathbf{Rad}(P'))^2}$. See Figure 1(c). Moreover, Definition 3 implies that $\mathbf{Rad}(P') \geq (1 - \epsilon^2)\mathbf{Rad}(P)$. Therefore, we have

$$\|o' - o\| \leq \sqrt{(\mathbf{Rad}(P))^2 - ((1 - \epsilon^2)\mathbf{Rad}(P))^2} \leq \sqrt{2\epsilon}\mathbf{Rad}(P). \quad (4)$$

► **Lemma 10.** *The distance $\|\tilde{o} - o'\| < (\sqrt{2}\epsilon + \sqrt{3}\epsilon')\mathbf{Rad}(P)$.*

Proof. Let L be the hyperplane orthogonal to the segment $\overline{\tilde{o}o'}$ and passing through the center o' . Suppose \tilde{o} is located on the left side of L . Then, there exists a point $b \in P'$ located on the right closed semi-sphere of $\mathbf{MEB}(P')$ divided by L (this result was proved in [18, 43] and see Lemma 2.2 in [18]). See Figure 1(d). That is, the angle $\angle bo'\tilde{o} \geq \pi/2$. As a consequence, we have

$$\|\tilde{o} - o'\| \leq \sqrt{\|\tilde{o} - b\|^2 - \|b - o'\|^2}. \quad (5)$$

Moreover, since $\|\tilde{o} - b\| \leq r \leq (1 + \epsilon'^2)\mathbf{Rad}(P)$ and $\|b - o'\| = \mathbf{Rad}(P') \geq (1 - \epsilon^2)\mathbf{Rad}(P)$, (5) implies that $\|\tilde{o} - o'\| \leq \sqrt{(1 + \epsilon'^2)^2 - (1 - \epsilon^2)^2}\mathbf{Rad}(P)$, where this upper bound is equal to

$$\sqrt{2\epsilon'^2 + \epsilon'^4 + 2\epsilon^2 - \epsilon^4}\mathbf{Rad}(P) < \sqrt{3\epsilon'^2 + 2\epsilon^2}\mathbf{Rad}(P) < (\sqrt{2}\epsilon + \sqrt{3}\epsilon')\mathbf{Rad}(P). \quad (6)$$

By triangle inequality, Lemmas 8 and 10, we immediately have

$$\|\tilde{o} - o\| \leq \|\tilde{o} - o'\| + \|o' - o\| < (2\sqrt{2}\epsilon + \sqrt{3}\epsilon')\mathbf{Rad}(P). \quad (7)$$

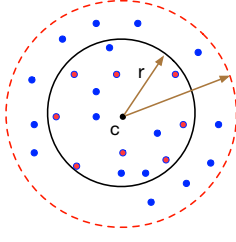
This completes the proof of Theorem 7.

5 Sublinear Time Algorithms for MEB under Stability Assumption

Suppose $\epsilon \in (0, 1)$. We assume that the given instance P is an (ϵ^2, β) -stable instance where β is larger than a given lower bound β_0 (i.e., $\beta > \beta_0$). Using Theorem 7, we present two different sublinear time sampling algorithms for computing MEB. Following most of the articles on sublinear time algorithms (e.g., [29, 64, 65]), in each sampling step of our algorithms, we always take the sample **independently and uniformly at random**.

5.1 The First Algorithm

The first algorithm is based on the theory of VC dimension and ϵ -nets [49, 81]. Roughly speaking, we compute an approximate MEB of a small random sample (say, $\mathbb{B}(c, r)$), and expand the ball slightly; then we prove that this expanded ball is an approximate MEB of the whole data set (see Figure 2). Our key idea is to show that $\mathbb{B}(c, r)$ covers at least $(1 - \beta_0)n$ points and therefore c is close to the optimal center by Theorem 7. As emphasized



■ **Figure 2** An illustration for the first sampling algorithm. The red points are the samples; we expand $\mathbb{B}(c, r)$ slightly and the larger ball is a radius-approximate MEB of the whole input point set.

in Section 1.1, our result is a single-criterion approximation. If simply applying the uniform sample idea without the stability assumption (as the ideas in [33, 53]), it will result in a bi-criteria approximation where the ball has to cover less than n points for achieving the desired bounded radius.

► **Theorem 11.** *With probability $1 - \eta$, Algorithm 1 returns a λ -radius approximate MEB of P , where*

$$\lambda = \frac{(1 + (2\sqrt{2} + \sqrt{3})\epsilon)(1 + \epsilon^2)}{1 - \epsilon^2} \quad (8)$$

and $\lambda = 1 + O(\epsilon)$ if ϵ is a fixed number in $(0, 1)$.

■ **Algorithm 1 MEB Algorithm I.**

Input: Two parameters $0 < \epsilon, \eta < 1$; an (ϵ^2, β) -stable instance P of MEB problem in \mathbb{R}^d , where β is larger than a known lower bound $\beta_0 > 0$.

- 1: Sample a set S of $\Theta(\frac{1}{\beta_0} \cdot \max\{\log \frac{1}{\eta}, d \log \frac{d}{\beta_0}\})$ points from P uniformly at random.
- 2: Apply any approximate MEB algorithm (such as the core-set based algorithm [16]) to compute a $(1 + \epsilon^2)$ -radius approximate MEB of S , and let the obtained ball be $\mathbb{B}(c, r)$.
- 3: Output the ball $\mathbb{B}(c, \frac{1 + (2\sqrt{2} + \sqrt{3})\epsilon}{1 - \epsilon^2} r)$.

Before proving Theorem 11, we prove the following lemma first.

► **Lemma 12.** *Let S be a set of $\Theta(\frac{1}{\beta_0} \cdot \max\{\log \frac{1}{\eta}, d \log \frac{d}{\beta_0}\})$ points sampled randomly and independently from a given point set $P \subset \mathbb{R}^d$, and B be any ball covering S . Then, with probability $1 - \eta$, $|B \cap P| \geq (1 - \beta_0)|P|$.*

Proof. Consider the range space $\Sigma = (P, \Phi)$ where each range $\phi \in \Phi$ is the complement of a ball in the space. In a range space, a subset $Y \subset P$ is a β_0 -net if for any $\phi \in \Phi$, $\frac{|Y \cap \phi|}{|Y|} \geq \beta_0 \implies Y \cap \phi \neq \emptyset$. Since $|S| = \Theta(\frac{1}{\beta_0} \cdot \max\{\log \frac{1}{\eta}, d \log \frac{d}{\beta_0}\})$, we know that S is a β_0 -net of P with probability $1 - \eta$ [49, 81]. Thus, if $|B \cap P| < (1 - \beta_0)|P|$, i.e., $|P \setminus B| > \beta_0|P|$, we have $S \cap (P \setminus B) \neq \emptyset$. This contradicts to the fact that S is covered by B . Consequently, $|B \cap P| \geq (1 - \beta_0)|P|$. ◀

Proof of Theorem 11. Denote by o the center of $\mathbf{MEB}(P)$. Since $S \subset P$ and $\mathbb{B}(c, r)$ is a $(1 + \epsilon^2)$ -radius approximate MEB of S , we know that $r \leq (1 + \epsilon^2)\mathbf{Rad}(P)$. Moreover, Lemma 12 implies that $|\mathbb{B}(c, r) \cap P| \geq (1 - \beta_0)|P|$ with probability $1 - \eta$. Suppose it is true and let $P' = \mathbb{B}(c, r) \cap P$. Then, we have the distance

$$\|c - o\| \leq (2\sqrt{2} + \sqrt{3})\epsilon \mathbf{Rad}(P) \quad (9)$$

via Theorem 7 (we set $\epsilon' = \epsilon$). For simplicity, we use x to denote $(2\sqrt{2} + \sqrt{3})\epsilon$. The inequality (9) implies that the point set P is covered by the ball $\mathbb{B}(c, (1+x)\mathbf{Rad}(P))$. Note that we cannot directly return $\mathbb{B}(c, (1+x)\mathbf{Rad}(P))$ as the final result, since we do not know the value of $\mathbf{Rad}(P)$. Thus, we have to estimate the radius $(1+x)\mathbf{Rad}(P)$.

Since P' is covered by $\mathbb{B}(c, r)$ and $|P'| \geq (1 - \beta_0)|P|$, r should be at least $(1 - \epsilon^2)\mathbf{Rad}(P)$ due to Definition 3. Hence, we have

$$\frac{1+x}{1-\epsilon^2}r \geq (1+x)\mathbf{Rad}(P). \quad (10)$$

That is, P is covered by the ball $\mathbb{B}(c, \frac{1+x}{1-\epsilon^2}r)$. Moreover, the radius

$$\frac{1+x}{1-\epsilon^2}r \leq \frac{1+x}{1-\epsilon^2}(1+\epsilon^2)\mathbf{Rad}(P). \quad (11)$$

This means that ball $\mathbb{B}(c, \frac{1+x}{1-\epsilon^2}r)$ is a λ -radius approximate MEB of P , where

$$\lambda = (1+\epsilon^2)\frac{1+x}{1-\epsilon^2} = \frac{(1+(2\sqrt{2}+\sqrt{3})\epsilon)(1+\epsilon^2)}{1-\epsilon^2} \quad (12)$$

and $\lambda = 1 + O(\epsilon)$ if ϵ is a fixed number in $(0, 1)$. \blacktriangleleft

Running time of Algorithm 1. For simplicity, we assume $\log \frac{1}{\eta} < d \log \frac{d}{\beta_0}$. If we use the core-set based algorithm [16] to compute $\mathbb{B}(c, r)$ (see Remark 5), the running time of Algorithm 1 is $O(\frac{1}{\epsilon^2}(|S|d + \frac{1}{\epsilon^6}d)) = O(\frac{d^2}{\epsilon^2\beta_0} \log \frac{d}{\beta_0} + \frac{d}{\epsilon^8}) = \tilde{O}(d^2)$ where the hidden factor depends on ϵ and β_0 .

► **Remark 13.** If the dimensionality d is too high, the random projection technique *Johnson-Lindenstrauss (JL) transform* [30] can be used to approximately preserve the radius of enclosing ball [2, 56, 79]. However, it is not very useful for reducing the time complexity of Algorithm 1. If we apply the JL-transform on the sampled $\Theta(\frac{d}{\beta_0} \log \frac{d}{\beta_0})$ points in Step 1, the JL-transform step itself already takes $\Omega(\frac{d^2}{\beta_0} \log \frac{d}{\beta_0})$ time.

5.2 The Second Algorithm

Our first algorithm in Section 5.1 is simple, but has a sample size (*i.e.*, the number of sampled points) depending on the dimensionality d , while **the second algorithm has a sample size independent of both n and d** (it is particularly important when a kernel function is applied, because the new dimension could be very large or even $+\infty$). We briefly overview our idea first.

High level idea of the second algorithm. Recall our Remark 5 (ii). If we know the value of $(1+\epsilon)\mathbf{Rad}(P)$, we can perform almost the same core-set construction procedure described in Theorem 4 to achieve an approximate center of $\mathbf{MEB}(P)$, where the only difference is that we add a point with distance at least $(1+\epsilon)\mathbf{Rad}(P)$ to o_i in each iteration. In this way, we avoid selecting the farthest point to o_i , since this operation will inevitably have a linear time complexity. To implement our strategy in sublinear time, we need to determine the value of $(1+\epsilon)\mathbf{Rad}(P)$ first. We propose Lemma 14 below to estimate the range of $\mathbf{Rad}(P)$, and then perform a binary search on the range to determine the value of $(1+\epsilon)\mathbf{Rad}(P)$ approximately. Based on the stability property, we observe that the core-set construction procedure can serve as an “oracle” to help us to guess the value of $(1+\epsilon)\mathbf{Rad}(P)$ (see Algorithm 3). Let $h > 0$ be a candidate. We add a point with distance at least h to o_i in each iteration. We

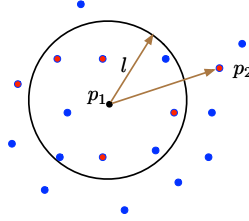
prove that the procedure cannot continue for more than z iterations if $h \geq (1 + \epsilon)\mathbf{Rad}(P)$, and will continue more than z iterations with constant probability if $h < (1 - \epsilon)\mathbf{Rad}(P)$, where z is the size of core-set described in Theorem 4. Also, during the core-set construction, we add the points to the core-set via random sampling, rather than a deterministic way. A minor issue here is that we need to replace ϵ by ϵ^2 in Theorem 4, so as to achieve the overall $(1 + O(\epsilon))$ -radius approximation in the following analysis. Below, we introduce Lemma 14 and Theorem 16 first, and then present the main result in Theorem 17.

► **Lemma 14.** *Given a parameter $\eta \in (0, 1)$, one selects an arbitrary point $p_1 \in P$ and takes a sample $Q \subset P$ with $|Q| = \frac{1}{\beta_0} \log \frac{1}{\eta}$ uniformly at random. Let $p_2 = \arg \max_{p \in Q} \|p - p_1\|$. Then, with probability $1 - \eta$,*

$$\mathbf{Rad}(P) \in [\frac{1}{2}\|p_1 - p_2\|, \frac{1}{1 - \epsilon^2}\|p_1 - p_2\|]. \quad (13)$$

Proof. First, the lower bound of $\mathbf{Rad}(P)$ is obvious since $\|p_1 - p_2\|$ is always no larger than $2\mathbf{Rad}(P)$. Then, we consider the upper bound. Let $\mathbb{B}(p_1, l)$ be the ball covering exactly $(1 - \beta_0)n$ points of P , and thus $l \geq (1 - \epsilon^2)\mathbf{Rad}(P)$ according to Definition 3. To complete our proof, we also need the following folklore lemma presented in [32].

► **Lemma 15** ([32]). *Let N be a set of elements, and N' be a subset of N with size $|N'| = \tau |N|$ for some $\tau \in (0, 1)$. Given from N , then with probability*



■ **Figure 3** An illustration of Lemma 14; the red points are the sampled set Q .

In Lemma 15, let N and N' be the point set P and the subset $P \setminus \mathbb{B}(p_1, l)$, respectively. We know that Q contains at least one point from N' according to Lemma 15 (by setting $\tau = \beta_0$). Namely, Q contains at least one point outside $\mathbb{B}(p_1, l)$. Moreover, because $p_2 = \arg \max_{p \in Q} \|p - p_1\|$, we have $\|p_1 - p_2\| \geq l \geq (1 - \epsilon^2)\mathbf{Rad}(P)$, i.e., $\mathbf{Rad}(P) \leq \frac{1}{1 - \epsilon^2}\|p_1 - p_2\|$ (see Figure 3 for an illustration). ◀

Lemma 14 immediately implies the following result.

► **Theorem 16.** *In Lemma 14, the ball $\mathbb{B}(p_1, \frac{2}{1 - \epsilon^2}\|p_1 - p_2\|)$ is a $\frac{4}{1 - \epsilon^2}$ -radius approximate MEB of P , with probability $1 - \eta$.*

Proof. From the upper bound in Lemma 14, we know that $\frac{2}{1 - \epsilon^2}\|p_1 - p_2\| \geq 2\mathbf{Rad}(P)$. Since $\|p_1 - p\| \leq 2\mathbf{Rad}(P)$ for any $p \in P$, the ball $\mathbb{B}(p_1, \frac{2}{1 - \epsilon^2}\|p_1 - p_2\|)$ covers the whole point set P . From the lower bound in Lemma 14, we know that $\frac{2}{1 - \epsilon^2}\|p_1 - p_2\| \leq \frac{4}{1 - \epsilon^2}\mathbf{Rad}(P)$. Therefore, it is a $\frac{4}{1 - \epsilon^2}$ -radius approximate MEB of P . ◀

Since $|Q| = \frac{1}{\beta_0} \log \frac{1}{\eta}$ in Lemma 14, Theorem 16 indicates that we can easily obtain a $\frac{4}{1 - \epsilon^2}$ -radius approximate MEB of P in $O(\frac{1}{\beta_0}(\log \frac{1}{\eta})d)$ time. Below, we present our second sampling algorithm (Algorithm 2) that can achieve a much lower $(1 + O(\epsilon))$ -approximation

ratio. Algorithm 3 serves as a subroutine in Algorithm 2. In Algorithm 3, we simply set $z = \frac{3}{\epsilon^2}$ with $s = 1/3$ as described in Theorem 4 (as mentioned before, we replace ϵ by ϵ^2); we compute o_i having distance less than $s \frac{\epsilon^2}{1+\epsilon^2} \mathbf{Rad}(T)$ to the center of $\mathbf{MEB}(T)$ in Step 2(1).

► **Theorem 17.** *With probability $1 - \eta_0$, Algorithm 2 returns a λ -radius approximate MEB of P , where*

$$\lambda = \frac{(1+x_1)(1+x_2)}{1+\epsilon^2} \quad \text{with} \quad x_1 = O\left(\frac{\epsilon^2}{1-\epsilon^2}\right), x_2 = O\left(\frac{\epsilon}{\sqrt{1-\epsilon^2}}\right), \quad (14)$$

and $\lambda = 1 + O(\epsilon)$ if ϵ is a fixed number in $(0, 1)$. The running time is $\tilde{O}((\frac{1}{\epsilon^2 \beta_0} + \frac{1}{\epsilon^8})d)$, where $\tilde{O}(f) = O(f \cdot \text{polylog}(\frac{1}{\epsilon}, \frac{1}{\eta_0}))$.

■ Algorithm 2 MEB Algorithm II.

Input: Two parameters $0 < \epsilon, \eta_0 < 1$; an (ϵ^2, β) -stable instance P of MEB problem in \mathbb{R}^d , where β is larger than a given lower bound $\beta_0 > 0$. Set the interval $[a, b]$ for $\mathbf{Rad}(P)$ that is obtained by Lemma 14.

- 1: Among the set $\{(1-\epsilon^2)a, (1+\epsilon^2)(1-\epsilon^2)a, \dots, (1+\epsilon^2)^w(1-\epsilon^2)a = (1+\epsilon^2)b\}$ where $w = \lceil \log_{1+\epsilon^2} \frac{2}{(1-\epsilon^2)^2} \rceil + 1 = O(\frac{1}{\epsilon^2})$, perform binary search for the value h by using Algorithm 3 with $z = \frac{3}{\epsilon^2}$ and $\eta = \frac{\eta_0}{2 \log w}$.
- 2: Suppose that Algorithm 3 returns “no” when $h = (1+\epsilon^2)^{i_0}(1-\epsilon^2)a$ and returns “yes” when $h = (1+\epsilon^2)^{i_0+1}(1-\epsilon^2)a$.
- 3: Run Algorithm 3 again with $h = (1+\epsilon^2)^{i_0+2}a$, $z = \frac{3}{\epsilon^2}$, and $\eta = \eta_0/2$; let \tilde{o} be the obtained ball center of T when the loop stops.
- 4: Return the ball $\mathbb{B}(\tilde{o}, r)$, where $r = \frac{1+(2\sqrt{2}+\frac{2\sqrt{6}}{\sqrt{1-\epsilon^2}})\epsilon}{1+\epsilon^2} h$.

■ Algorithm 3 Oracle for testing h .

Input: An instance P , a parameter $\eta \in (0, 1)$, $h > 0$, and a positive integer z .

- 1: Initially, arbitrarily select a point $p \in P$ and let $T = \{p\}$.
- 2: $i = 1$; repeat the following steps:
 - (1) Compute an approximate MEB of T and let the ball center be o_i as described in Theorem 4 (replace ϵ by ϵ^2 and set $s = 1/3$).
 - (2) Sample a set $Q \subset P$ with $|Q| = \frac{1}{\beta_0} \log \frac{z}{\eta}$ uniformly at random.
 - (3) Select the point $q \in Q$ that is farthest to o_i , and add it to T .
 - (4) If $\|q - o_i\| < h$, stop the loop and output “yes”.
 - (5) $i = i + 1$; if $i > z$, stop the loop and output “no”.

Before proving Theorem 17, we provide Lemma 18 first.

► **Lemma 18.** *If $h \geq (1+\epsilon^2)\mathbf{Rad}(P)$, Algorithm 3 returns “yes”; else if $h < (1-\epsilon^2)\mathbf{Rad}(P)$, Algorithm 3 returns “no” with probability at least $1 - \eta$.*

Proof. First, we assume that $h \geq (1+\epsilon^2)\mathbf{Rad}(P)$. Recall the remark following Theorem 4. If we always add a point q with distance at least $h \geq (1+\epsilon^2)\mathbf{Rad}(P)$ to o_i , the loop 2(1)-(5) cannot continue more than z iterations, i.e., Algorithm 3 will return “yes”.

Now, we consider the case $h < (1-\epsilon^2)\mathbf{Rad}(P)$. Similar to the proof of Lemma 14, we consider the ball $\mathbb{B}(o_i, l)$ covering exactly $(1-\beta_0)n$ points of P . According to Definition 3, we know that $l \geq (1-\epsilon^2)\mathbf{Rad}(P) > h$. Also, with probability $1 - \eta/z$, the sample Q contains

38:14 Stability Yields Sublinear Time Algorithms

at least one point outside $\mathbb{B}(o_i, l)$ due to Lemma 15. By taking the union bound, with probability $(1 - \eta/z)^z \geq 1 - \eta$, $\|q - o_i\|$ is always larger than h and eventually Algorithm 3 will return “no”. \blacktriangleleft

Proof of Theorem 17. Since Algorithm 3 returns “no” when $h = (1 + \epsilon^2)^{i_0}(1 - \epsilon^2)a$ and returns “yes” when $h = (1 + \epsilon^2)^{i_0+1}(1 - \epsilon^2)a$, from Lemma 18 we know that

$$(1 + \epsilon^2)^{i_0}(1 - \epsilon^2)a < (1 + \epsilon^2)\mathbf{Rad}(P); \quad (15)$$

$$(1 + \epsilon^2)^{i_0+1}(1 - \epsilon^2)a \geq (1 - \epsilon^2)\mathbf{Rad}(P). \quad (16)$$

The above inequalities together imply that

$$\frac{(1 + \epsilon^2)^3}{1 - \epsilon^2}\mathbf{Rad}(P) > (1 + \epsilon^2)^{i_0+2}a \geq (1 + \epsilon^2)\mathbf{Rad}(P). \quad (17)$$

Thus, when running Algorithm 3 with $h = (1 + \epsilon^2)^{i_0+2}a$ in Step 3, the algorithm returns “yes” (by the right hand-side of (17)). Then, consider the ball $\mathbb{B}(\tilde{o}, h)$. We claim that $|P \setminus \mathbb{B}(\tilde{o}, h)| < \beta_0 n$. Otherwise, the sample Q contains at least one point outside $\mathbb{B}(\tilde{o}, h)$ with probability $1 - \eta/z$ in Step 2(2) of Algorithm 3, *i.e.*, the loop will continue. Thus, it contradicts to the fact that the algorithm returns “yes”. Let $P' = P \cap \mathbb{B}(\tilde{o}, h)$, and then $|P'| \geq (1 - \beta_0)n$. Moreover, the left hand-side of (17) indicates that

$$h = (1 + \epsilon^2)^{i_0+2}a < (1 + \frac{8\epsilon^2}{1 - \epsilon^2})\mathbf{Rad}(P). \quad (18)$$

Now, we can apply Theorem 7, where we set “ ϵ' ” to be “ $\sqrt{\frac{8\epsilon^2}{1 - \epsilon^2}}$ ” in the theorem. Let o be the center of $\mathbf{MEB}(P)$. Consequently, we have

$$\|\tilde{o} - o\| < (2\sqrt{2} + 2\sqrt{6}/\sqrt{1 - \epsilon^2})\epsilon \cdot \mathbf{Rad}(P). \quad (19)$$

For simplicity, we let $x_1 = \frac{8\epsilon^2}{1 - \epsilon^2}$ and $x_2 = (2\sqrt{2} + 2\sqrt{6}/\sqrt{1 - \epsilon^2})\epsilon$. Hence, $h \leq (1 + x_1)\mathbf{Rad}(P)$ and $\|\tilde{o} - o\| \leq x_2\mathbf{Rad}(P)$ in (18) and (19). From (19), we know that $P \subset \mathbb{B}(\tilde{o}, (1 + x_2)\mathbf{Rad}(P))$. From the right hand-side of (17), we know that $(1 + x_2)\mathbf{Rad}(P) \leq \frac{1+x_2}{1+\epsilon^2}h$. Thus, we have $P \subset \mathbb{B}(\tilde{o}, \frac{1+x_2}{1+\epsilon^2}h)$ where $\frac{1+x_2}{1+\epsilon^2}h = \frac{1+(2\sqrt{2} + \frac{2\sqrt{6}}{\sqrt{1-\epsilon^2}})\epsilon}{1+\epsilon^2}h$. Also, the radius

$$\frac{1 + x_2}{1 + \epsilon^2}h \underbrace{\leq}_{\text{by (18)}} \frac{(1 + x_2)(1 + x_1)}{1 + \epsilon^2}\mathbf{Rad}(P) = \lambda \cdot \mathbf{Rad}(P). \quad (20)$$

Thus $\mathbb{B}(\tilde{o}, \frac{1+x_2}{1+\epsilon^2}h)$ is a λ -radius approximate MEB of P , and $\lambda = 1 + O(\epsilon)$ if ϵ is fixed.

Success probability. The success probability of Algorithm 3 is $1 - \eta$. In Algorithm 2, we set $\eta = \frac{\eta_0}{2 \log w}$ in Step 1 and $\eta = \eta_0/2$ in Step 3, respectively. We take the union bound and the success probability of Algorithm 2 is $(1 - \frac{\eta_0}{2 \log w})^{\log w} (1 - \eta_0/2) > 1 - \eta_0$.

Running time. As the subroutine, Algorithm 3 runs in $O(z(\frac{1}{\beta_0}(\log \frac{z}{\eta})d + \frac{1}{\epsilon^6}d))$ time; Algorithm 2 calls the subroutine $O(\log(\frac{1}{\epsilon^2}))$ times. Note that $z = O(\frac{1}{\epsilon^2})$. Thus, the total running time is $\tilde{O}((\frac{1}{\epsilon^2 \beta_0} + \frac{1}{\epsilon^8})d)$. \blacktriangleleft

6 Extensions

We also present two important extensions in this paper (due to the space limit, we place the details to our full paper). We briefly introduce the main ideas and summarize the results below.

We first consider MEB with outliers under the stability assumption and provide a sublinear time constant factor radius approximation. We also consider the general case without the stability assumption. An interesting observation is that the ideas developed for stable instance can even help us to develop a hybrid approach for MEB (without or with outliers) when the stability assumption does not hold. First, we “suppose” the input instance is (α, β) -stable where “ α ” and “ β ” are carefully designed based on the pre-specified radius error bound ϵ and covering error bound δ , and compute a “potential” $(1 + \epsilon)$ -radius approximation (say a ball B_1); then we apply the recently proposed sublinear time bi-criteria MEB with outliers algorithm [31] to compute a “potential” $(1 - \delta)$ -covering approximation (say a ball B_2); finally, we determine the final output based on the ratio of their radii. Specifically, we set a threshold τ that is determined by the given radius error bound ϵ . If the ratio is no larger than τ , we know that B_1 is a “true” $(1 + \epsilon)$ -radius approximation and return it; otherwise, we return B_2 that is a “true” $(1 - \delta)$ -covering approximation. Moreover, for the latter case (*i.e.*, returning a $(1 - \delta)$ -covering approximation), we will show that our proposed algorithm yields a radius not only being strictly smaller than $\mathbf{Rad}(P)$, but also having a gap of $\Theta(\epsilon^2) \cdot \mathbf{Rad}(P)$ to $\mathbf{Rad}(P)$ (*i.e.*, the returned radius is at most $(1 - \Theta(\epsilon^2)) \cdot \mathbf{Rad}(P)$). Our algorithm only needs uniform sampling and a single pass over the input data, where the space complexity in memory is $O(d)$ (the hidden factor depends on ϵ and δ); if the input data matrix is sparse (*i.e.*, $M = o(nd)$), the time complexity is sublinear. Furthermore, we propose the similar results for the polytope distance and SVM problems (for both stable instance and general instance).

7 Future Work

Following our work, several interesting problems deserve to be studied in future. For example, different from radius approximation, the current research on covering approximation of MEB is still inadequate. In particular, can we provide a lower bound for the complexity of computing covering approximate MEB, as the lower bound result for radius approximate MEB proved by [25]? Also, is it possible to extend the stability notion to other geometric optimization problems with more complicated structures (like subspace fitting and clustering [36], and regression problems [67])?

References

- 1 Pankaj K. Agarwal, Sarel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52:1–30, 2005.
- 2 Pankaj K. Agarwal, Sarel Har-Peled, and Hai Yu. Embeddings of surfaces, curves, and moving points in euclidean space. In *Proceedings of the 23rd ACM Symposium on Computational Geometry, Gyeongju, South Korea, June 6-8, 2007*, pages 381–389, 2007.
- 3 Pankaj K. Agarwal, Sarel Har-Peled, and Hai Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1-3):38–58, 2008.
- 4 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015.
- 5 Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of algorithms*, 12(1):38–56, 1991.

- 6 Zeyuan Allen Zhu, Zhenyu Liao, and Yang Yuan. Optimization algorithms for faster computational geometry. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 53:1–53:6, 2016.
- 7 Noga Alon, Seannie Dar, Michal Parnas, and Dana Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3):393–417, 2003.
- 8 Pranjali Awasthi, Avrim Blum, and Or Sheffet. Stability yields a PTAS for k-median and k-means clustering. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 309–318, 2010.
- 9 Pranjali Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Inf. Process. Lett.*, 112(1-2):49–54, 2012.
- 10 Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Clustering under approximation stability. *Journal of the ACM (JACM)*, 60(2):8, 2013.
- 11 Maria-Florina Balcan and Mark Braverman. Finding low error clusterings. In *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*, 2009.
- 12 Maria-Florina Balcan, Nika Haghtalab, and Colin White. k-center clustering under perturbation resilience. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 68:1–68:14, 2016.
- 13 Maria-Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *SIAM J. Comput.*, 45(1):102–155, 2016.
- 14 Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Oper. Res.*, 52(1):35–53, 2004.
- 15 Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability & Computing*, 21(5):643–660, 2012.
- 16 Mihai Bădoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802, 2003.
- 17 Mihai Bădoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22, 2008.
- 18 Mihai Bădoiu, Sarel 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.
- 19 Giuseppe Carlo Calafiore and Marco C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Math. Program.*, 102(1):25–46, 2005.
- 20 Matteo Ceccarelo, Andrea Pietracaprina, and Geppino Pucci. Solving k-center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially. *PVLDB*, 12(7):766–778, 2019.
- 21 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom.*, 47(2):240–247, 2014.
- 22 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.
- 23 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.
- 24 Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):63, 2010.
- 25 Kenneth L. Clarkson, Elad Hazan, and David P. Woodruff. Sublinear optimization for machine learning. *J. ACM*, 59(5):23:1–23:49, 2012.
- 26 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273, 1995.
- 27 David J. Crisp and Christopher J. C. Burges. A geometric interpretation of v-SVM classifiers. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *NIPS*, pages 244–250. The MIT Press, 1999.
- 28 Artur Czumaj and Christian Sohler. Sublinear-time algorithms. *Survey paper*, 2004.

- 29 Artur Czumaj and Christian Sohler. Sublinear-time approximation for clustering via random sampling. In *International Colloquium on Automata, Languages, and Programming*, pages 396–407. Springer, 2004.
- 30 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.
- 31 Hu Ding. A sub-linear time framework for geometric optimization with outliers in high dimensions. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 38:1–38:21, 2020.
- 32 Hu Ding and Jinhui Xu. Sub-linear time hybrid approximations for least trimmed squares estimator and related problems. In *Proceedings of the International Symposium on Computational geometry (SoCG)*, page 110, 2014.
- 33 Hu Ding, Haikuo Yu, and Zixiu Wang. Greedy strategy works for k-center clustering with outliers and coresets construction. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 40:1–40:16, 2019.
- 34 Alon Efrat, Micha Sharir, and Alon Ziv. Computing the smallest k-enclosing circle and related problems. *Computational Geometry*, 4(3):119–136, 1994.
- 35 Dan Feldman. Core-sets: An updated survey. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 10(1), 2020.
- 36 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578. ACM, 2011.
- 37 Dan Feldman, Chongyuan Xiang, Ruihao Zhu, and Daniela Rus. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2017, Pittsburgh, PA, USA, April 18-21, 2017*, pages 3–15, 2017.
- 38 Kaspar Fischer, Bernd Gärtner, and Martin Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, pages 630–641, 2003.
- 39 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.
- 40 Dan Garber and Elad Hazan. Approximating semidefinite programs in sublinear time. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1080–1088, 2011.
- 41 Bernd Gärtner and Martin Jaggi. Coresets for polytope distance. In *Proceedings of the International Symposium on Computational geometry (SoCG)*, pages 33–42, 2009.
- 42 Elmer G. Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 4(1):61–80, 1966.
- 43 Ashish Goel, Piotr Indyk, and Kasturi R Varadarajan. Reductions among high dimensional proximity problems. In *SODA*, volume 1, pages 769–778. Citeseer, 2001.
- 44 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.
- 45 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 46 Laszlo Gyongyosi and Sandor Imre. Geometrical analysis of physically allowed quantum cloning transformations for quantum cryptography. *Information Sciences*, 285:1–23, 2014.
- 47 Sarel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

- 48 Sarel Har-Peled and Yusu Wang. Shape fitting with outliers. *SIAM Journal on Computing*, 33(2):269–285, 2004.
- 49 David Haussler and Emo Welzl. eps-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.
- 50 Kohei Hayashi and Yuichi Yoshida. Minimizing quadratic functions in constant time. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2217–2225, 2016.
- 51 Elad Hazan, Tomer Koren, and Nati Srebro. Beating SGD: learning svms in sublinear time. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1233–1241, 2011.
- 52 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.
- 53 Lingxiao Huang, Shaofeng Jiang, Jian Li, and Xuan Wu. Epsilon-coresets for clustering (with outliers) in doubling metrics. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 814–825, 2018.
- 54 Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 428–434, 1999.
- 55 Piotr Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 154–159, 1999.
- 56 Michael Kerber and Sharath Raghvendra. Approximation and streaming algorithms for projective clustering via random projections. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, 2015.
- 57 Michael Kerber and R. Sharathkumar. Approximate čech complex in low and high dimensions. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 666–676, 2013.
- 58 Amer Krivosija and Alexander Munteanu. Probabilistic smallest enclosing ball in high dimensions via subgradient sampling. In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, volume 129 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 59 Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 299–308. IEEE, 2010.
- 60 Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, 8, 2003.
- 61 Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- 62 Jiří Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53(4):217–221, 1995.
- 63 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.
- 64 Adam Meyerson, Liadan O’callaghan, and Serge Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1-3):61–87, 2004.

- 65 Nina Mishra, Dan Oblinger, and Leonard Pitt. Sublinear time approximate clustering. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 439–447. Society for Industrial and Applied Mathematics, 2001.
- 66 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, USA, 1995.
- 67 David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. On the least trimmed squares estimator. *Algorithmica*, 69(1):148–183, 2014.
- 68 Frank Nielsen and Richard Nock. Approximating smallest enclosing balls with applications to machine learning. *Int. J. Comput. Geom. Appl.*, 19(5):389–414, 2009.
- 69 Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Locating a small cluster privately. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 413–427, 2016.
- 70 Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM (JACM)*, 59(6):28, 2012.
- 71 Rina Panigrahy. Minimum enclosing polytope in high dimensions. *arXiv preprint cs/0407020*, 2004.
- 72 Jeff M. Phillips. Coresets and sketches. *Computing Research Repository*, 2016.
- 73 J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.
- 74 Tim Roughgarden. Beyond worst-case analysis. *Commun. ACM*, 62(3):88–96, 2019.
- 75 Ronitt Rubinfeld. Sublinear time algorithms. *Citeseer*, 2006.
- 76 Ankan Saha, S. V. N. Vishwanathan, and Xinhua Zhang. New approximation algorithms for minimum enclosing convex shapes. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1146–1160, 2011.
- 77 B. Scholkopf, A. J. Smola, K. R. Muller, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.
- 78 Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002.
- 79 Donald R. Sheehy. The persistent homology of distance functions under random projection. In *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 328, 2014.
- 80 Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.
- 81 Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.
- 82 Hamid Zarrabi-Zadeh and Asish Mukhopadhyay. Streaming 1-center with outliers in high dimensions. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 83–86, 2009.

Data Structures Lower Bounds and Popular Conjectures

Pavel Dvořák ✉

Charles University, Prague, Czech Republic

Michal Koucký ✉

Charles University, Prague, Czech Republic

Karel Král ✉

Charles University, Prague, Czech Republic

Veronika Slívová ✉

Charles University, Prague, Czech Republic

Abstract

In this paper, we investigate the relative power of several conjectures that attracted recently lot of interest. We establish a connection between the Network Coding Conjecture (*NCC*) of Li and Li [25] and several data structure problems such as non-adaptive function inversion of Hellman [19] and the well-studied problem of polynomial evaluation and interpolation. In turn these data structure problems imply super-linear circuit lower bounds for explicit functions such as integer sorting and multi-point polynomial evaluation.

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

Keywords and phrases Data structures, Circuits, Lower bounds, Network Coding Conjecture

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.39

Related Version *Full Version*: <https://arxiv.org/abs/2102.09294>

Funding The authors were partially supported by Czech Science Foundation GAČR grant #19-27871X. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 823748. The fourth author was supported by Charles University project UNCE/SCI/004.

Acknowledgements We would like to thank to Mike Saks and Sagnik Mukhopadhyay for insightful discussions.

1 Introduction

One of the central problems in theoretical computer science is proving lower bounds in various models of computation such as circuits and data structures. Proving super-linear size lower bounds for circuits even when their depth is restricted is rather elusive. Similarly, proving polynomial lower bounds on query time for certain static data structure problems seems out of reach. To deal with this situation researchers developed various conjectures which if true would imply the sought after lower bounds. In this paper, we investigate the relative power of some of those conjectures. The Network Coding Conjecture (*NCC*) of Li and Li [25] attracted recently lot of attention and it was used to prove various lower bounds such as lower bounds on the size of circuits computing multiplication [3] and the number of IO operations needed for external memory sorting [13].

Another problem that is popular in cryptography is a certain data structure type problem for function inversion [19]. Corrigan-Gibbs and Kogan [9] observed that lower bounds for the function inversion problem imply lower bounds for logarithmic depth circuits. A similar more



© Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 39; pp. 39:1–39:15

Leibniz International Proceedings in Informatics



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

general observation was made by Viola [35]. In this paper, we establish a new connection between the function inversion problem and NCC. We show that NCC implies certain weak lower bounds for the inversion data structure problem. That in turn implies the same type of circuit lower bounds as given by Corrigan-Gibbs and Kogan [9]. We show that similar results apply to a host of other data structure problems such as the well-studied polynomial evaluation problem or the Finite Field Fourier transform problem. Corrigan-Gibbs and Kogan [9] gave their circuit lower bound for certain apriori undetermined function. We establish the same circuit lower bounds for sorting integers which is a very explicit function. Similarly, we establish a connection between data structure for polynomial evaluation and circuits for multi-point polynomial evaluation. Our results sharpen and generalize the picture emerging in the literature.

The data structures considered in this paper are static, non-adaptive, and systematic, i.e., a very restricted class of data structures for which lower bounds should perhaps be easier to obtain. Such data structure problems have the following structure: Given the input data described by N bits, create a data structure of size s . Then we receive a single query from a set of permissible queries and we are supposed to answer the query while non-adaptively inspecting at most t locations in the data structure and in the original data. The non-adaptivity means that the inspected locations are chosen only based on the query being answered but not on the content of the inspected memory. We show that when $s \geq \omega(N/\log \log N)$, polynomial lower bounds on t for certain problems would imply super-linear lower bounds on log-depth circuits for computing sorting, multi-point polynomial evaluation, and other problems.

We show that logarithmic lower bounds on t for the data structures can be derived from NCC even in the more generous setting of $s \geq \varepsilon N$ and when inspecting locations in the data structure is for free. This matches the lower bounds of Afshani [3] for certain circuit parameters derived from NCC. One can recover the same type of result they showed from our connection between NCC, data structure lower bounds, and circuit lower bounds. In this regards, NCC seems to be a stronger assumption than that certain functions require large boolean circuits or inefficient data structures. One would hope that for the strongly restricted data structure problems, obtaining the required lower bounds should be within our reach.

Our technique seems applicable to data structure problems that are *involutions* that are inverses of themselves. Although we use a lot of the same technical machinery as the previous papers on NCC our proofs involve new ideas. An interesting aspect of our proofs is that they apply the hypothesized data structure twice in the reductions. This is reminiscent of many quantum algorithms that use Hadamard transform twice in a row.

Organization. The statement of our main results is in Section 4. In the next section, we review the data structure problems we consider. Then we provide a precise definition of Network Coding Conjecture in Section 3. In Section 5 we prove our main result for function inversion. In Section 6 we discuss the connection between data structure and circuit lower bounds for explicit functions. Some of the proofs are left for the full version [12].

2 Data Structure Problems

In this paper, we study lower bounds on *systematic data structures* for various problems – function inversion, polynomial evaluation, and polynomial interpolation. We are given an input $I = \{x_0, \dots, x_{n-1}\}$, where each $x_i \in [n] = \{0, \dots, n-1\}$ or each x_i is an element of

some field \mathbb{F} . First, a data structure algorithm can preprocess I to produce an advice string \mathbf{a}_I of s bits (we refer to the parameter s as *space* of the data structure \mathcal{D}). Then, we are given a query q and the data structure should produce a correct answer (what is a correct answer depends on the problem). To answer a query q , the data structure \mathcal{D} has access to the whole advice string \mathbf{a}_I and can make t probes to the input I , i.e., read at most t elements from I . We refer to the parameter t as the query time of the data structure.

We consider non-uniform data structures as we want to provide connections between data structures and non-uniform circuits. Formally, a non-uniform systematic data structure \mathcal{D}_n for an input $I = \{x_0, \dots, x_{n-1}\}$ is a pair of algorithms $(\mathcal{P}_n, \mathcal{Q}_n)$ with oracle access to I . The algorithm \mathcal{P}_n produces the advice string $\mathbf{a}_I \in \{0, 1\}^s$. The algorithm \mathcal{Q}_n with inputs \mathbf{a}_I and a query q outputs a correct answer to the query q with at most t oracle probes to I . The algorithms \mathcal{P}_n and \mathcal{Q}_n can differ for each $n \in \mathbb{N}$.

2.1 Function Inversion

In the function inversion problem, we are given a function $f : [n] \rightarrow [n]$ and a point $y \in [n]$ and we want to find $x \in [n]$ such that $f(x) = y$. This is a central problem in cryptography as many cryptographic primitives rely on the existence of a function that is hard to invert. To sum up we are interested in the following problem.

Function Inversion

<i>Input:</i>	A function $f : [n] \rightarrow [n]$ as an oracle.
<i>Preprocessing:</i>	Using f , prepare an advice string $\mathbf{a}_f \in \{0, 1\}^s$.
<i>Query:</i>	Point $y \in [n]$.
<i>Answer:</i>	Compute the value $f^{-1}(y)$, with a full access to \mathbf{a}_f and using at most t probes to the oracle for f .

We want to design an efficient data structure, i.e., make s and t as small as possible. There are two trivial solutions. The first one is that the whole function f^{-1} is stored in the advice string \mathbf{a}_f , thus $s = O(n \log n)$ and $t = 0$. The second one is that the whole function f is probed during answering a query $y \in [n]$, thus $t = O(n)$ and $s = 0$. Note that the space s of the data structure is the length of the advice string \mathbf{a}_f in bits but with one oracle-probe x_i the data structure reads the whole $f(x_i)$, thus with n oracle-probes we read the whole description of f , i.e., $n \log n$ bits.

The question is whether we can design a data structure with $s, t \leq o(n)$. Hellman [19] gave the first non-trivial solution and introduced a randomized (adaptive) systematic data structure which inverts a function with a constant probability (over the uniform choice of the function f and the query $y \in [n]$) and $s = O(n^{2/3} \log n)$ and $t = O(n^{2/3} \log n)$. Fiat and Naor [14] improved the result and introduced a data structure that inverts any function at any point, however with a slightly worse trade-off: $s^3 t = O(n^3 \log n)$. Hellman [19] also introduced a more efficient data structure for inverting a permutation – it inverts any permutation at any point and $st = O(n \log n)$. Thus, it seems that inverting a permutation is an easier problem than inverting an arbitrary function.

In this paper, we are interested in lower bounds for the inversion problem. Yao [36] gave a lower bound that any systematic data structure for the inversion problem must have $st \geq \Omega(n \log n)$, however, the lower bound is applicable only if $t \leq O(\sqrt{n})$. Since then, only slight progress was made. De et al. [10] improved the lower bound of Yao [36] to be applicable for the full range of t . Abusalah et al. [1] improved the trade-off, that for any k it must hold that $s^k t \geq \Omega(n^k)$. Seemingly, their result contradicts Hellman's trade-off ($s = t = O(n^{2/3} \log n)$) as it implies $s = t \geq n^{k/k+1}$ for any k . However, for Hellman's attack [19] we need that the function can be efficiently evaluated and the functions introduced by Abusalah et al. [1] cannot be efficiently evaluated. There is also a series of

papers [17, 30, 11, 8] that study how the probability of successful inversion depends on the parameters s and t . However, none of these results yields a better lower bound than $st \geq \Omega(n \log n)$. Hellman's trade-off is still the best-known upper bound trade-off for the inversion problem. Thus, there is still a substantial gap between the lower and upper bounds.

Another caveat of all known data structures for the inversion is that they heavily use adaptivity during answering queries $y \in [n]$. I.e., probes to the oracle depend on the advice string \mathbf{a} and answers to the oracle probes which have been already made. We are interested in non-adaptive data structures. We say a systematic data structure is *non-adaptive* if all oracle probes depend only on the query $y \in [n]$.

As non-adaptive data structures are weaker than adaptive ones, there is a hope that for non-adaptive data structures we could prove stronger lower bounds. Moreover, the non-adaptive data structure corresponds to circuits computation [31, 32, 34, 9]. Thus, we can derive a circuit lower bound from a strong lower bound for a non-adaptive data structure. Non-adaptive data structures were considered by Corrigan-Gibbs and Kogan [9]. They proved that improvement by a polynomial factor of Yao's lower bound [36] for non-adaptive data structures would imply the existence of a function $F : \{0, 1\}^N \rightarrow \{0, 1\}^N$ for $N = n \log n$ that cannot be computed by a linear-size and logarithmic-depth circuit. More formally, they prove that if a function $f : [n] \rightarrow [n]$ cannot be inverted by a non-adaptive data structure of space $O(n \log n / \log \log n)$ and query time $O(n^\varepsilon)$ for some $\varepsilon > 0$ then there exists a function $F : \{0, 1\}^N \rightarrow \{0, 1\}^N$ that cannot be computed by any circuit of size $O(N)$ and depth $O(\log N)$. They interpret $r \in \{0, 1\}^N$ as n numbers in $[n]$, i.e., $r = (r_1, \dots, r_n) \in \{0, 1\}^N$ where each $r_i \in [n]$. The function F is defined as $F(y) = F(y_1, \dots, y_n) = (f^{-1}(y_1), \dots, f^{-1}(y_n))$ where $f^{-1}(y_i) = \min\{x \in [n] \mid f(x) = y_i\}$ and $\min \emptyset = 0$. Informally, if the function f is hard to invert at some points, then it is hard to invert at all points together. Compared to the result of Corrigan-Gibbs and Kogan [9], we provide an explicit function (sorting n -bit integers) which will require large circuits if any of the functions f is hard to invert. A connection similar to Corrigan-Gibbs and Kogan between data structures and circuits was made also by Viola [35].

2.2 Evaluation and Interpolation of Polynomials

In this section, we describe two natural problems connected to polynomials. We consider our problems over a finite field \mathbb{F} to avoid issues with encoding reals.

Polynomial Evaluation over \mathbb{F}

<i>Input:</i>	Coefficients of a polynomial $p \in \mathbb{F}[x]$: $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F}$ (i.e., $p(x) = \sum_{i \in [n]} \alpha_i x^i$)
<i>Preprocessing:</i>	Using the input, prepare an advice string $\mathbf{a}_p \in \{0, 1\}^s$.
<i>Query:</i>	A number $x \in \mathbb{F}$.
<i>Answer:</i>	Compute the value $p(x)$, with a full access to \mathbf{a}_p and using at most t probes to the coefficients of p .

Polynomial Interpolation over \mathbb{F}

<i>Input:</i>	Point-value pairs of a polynomial $p \in \mathbb{F}[x]$ of degree at most $n - 1$: $(x_0, p(x_0)), \dots, (x_{n-1}, p(x_{n-1})) \in \mathbb{F} \times \mathbb{F}$ where $x_i \neq x_j$ for any two indices $i \neq j$
<i>Preprocessing:</i>	Using the input, prepare an advice string $\mathbf{a}_p \in \{0, 1\}^s$.
<i>Query:</i>	An index $j \in [n]$.
<i>Answer:</i>	Compute j -th coefficient of the polynomial p , i.e., the coefficient of x^j in p , with a full access to \mathbf{a}_p and using at most t probes to the oracle for point-value pairs.

In the paper we often use a version of polynomial interpolation where the points x_0, x_1, \dots, x_{n-1} are fixed in advance and the input consists just of $p(x_0), p(x_1), \dots, p(x_{n-1})$. Since we are interested in lower bounds, this makes our results slightly stronger.

Let $\mathbb{F} = \text{GF}(p^k)$ denote the Galois Field of p^k elements. Let n be a divisor of $p^k - 1$. It is a well-known fact that for any finite field \mathbb{F} its multiplicative group \mathbb{F}^* is cyclic (see e.g. Serre [28]). Thus, there is a primitive n -th root of unity $\sigma \in \mathbb{F}$ (that is an element σ such that $\sigma^n = 1$ and for each $1 \leq j < n$, $\sigma^j \neq 1$). Pollard [27] defines the *Finite Field Fourier transform* (FFFT) (with respect to σ) as a linear function $\text{FFFT}_{n,\sigma}: \mathbb{F}^n \rightarrow \mathbb{F}^n$ which satisfies:

$$\begin{aligned} \text{FFFT}_{n,\sigma}(\alpha_0, \dots, \alpha_{n-1}) &= (\beta_0, \dots, \beta_{n-1}) \text{ where} \\ \beta_i &= \sum_{j \in [n]} \alpha_j \sigma^{ij} \end{aligned} \quad \text{for any } i \in [n]$$

The inversion $\text{FFFT}_{n,\sigma}^{-1}$ is given by:

$$\begin{aligned} \text{FFFT}_{n,\sigma}^{-1}(\beta_0, \dots, \beta_{n-1}) &= (\alpha_0, \dots, \alpha_{n-1}) \text{ where} \\ \alpha_i &= \frac{1}{n} \sum_{j \in [n]} \beta_j \sigma^{-ij} \end{aligned} \quad \text{for any } i \in [n]$$

Here, $\frac{1}{n} = (\sum_{i=1}^n 1)^{-1}$ over \mathbb{F} . In our theorems we always set n to be a divisor of $|\mathbb{F}| - 1 = p^k - 1$ thus n modulo p is non-zero and the inverse exists. Observe, that $\text{FFFT}_{n,\sigma}^{-1} = \frac{1}{n} \text{FFFT}_{n,\sigma^{-1}}$.

FFFT is the finite field analog of Discrete Fourier transform (DFT) which works over complex numbers. The FFT algorithm by Cooley and Tukey [7] can be used for the case of finite fields as well (as observed by Pollard [27]) to get an algorithm using $O(n \log n)$ field operations (addition or multiplication of two numbers). Thus we can compute $\text{FFFT}_{n,\sigma}$ and its inverse in $O(n \log n)$ field operations.

It is easy to see that $\text{FFFT}_{n,\sigma}$ is actually evaluation of a polynomial in multiple special points (specifically in $\sigma^0, \dots, \sigma^{n-1}$). We can also see that it is a special case of interpolation by a polynomial in multiple special points since $\text{FFFT}_{n,\sigma}^{-1} = \frac{1}{n} \text{FFFT}_{n,\sigma^{-1}}$. We provide an NCC-based lower bound for data structures computing the polynomial evaluation. However, we use the data structure only for evaluating a polynomial in powers of a primitive root of unity. Thus, the same proof yields a lower bound for data structures computing the polynomial interpolation.

There is a great interest in data structures for polynomial evaluation in a cell probe model. In this model, some representation of a polynomial $p = \sum_{i \in [n]} \alpha_i x^i \in \mathbb{F}[x]$ is stored in a table \mathcal{T} of s_{cell} cells, each of w bits. Usually, w is set to $O(\log |\mathbb{F}|)$, that we can store an element of \mathbb{F} in a single cell. On a query $x \in \mathbb{F}$ the data structure should output $p(x)$ making at most t_{cell} probes to the table \mathcal{T} . A difference between data structures in the cell probe model and systematic data structures is that a data structure in the cell probe model is charged for any probe to the table \mathcal{T} but a systematic data structure is charged only for probes to the input (the coefficients α_i), reading from the advice string \mathbf{a}_p is for free. Note that, the coefficients α_i of p do not have to be even stored in the table \mathcal{T} . There are again two trivial solutions. The first one is that we store a value $p(x)$ for each $x \in \mathbb{F}$ and on a query $x \in \mathbb{F}$ we probe just one cell. Thus, we would get $t_{\text{cell}} = 1$ and $s_{\text{cell}} = |\mathbb{F}|$ (we assume that we can store an element of \mathbb{F} in a single cell). The second one is that we store the coefficients of p and on a query $x \in \mathbb{F}$ we probe all cells and compute the value $p(x)$. Thus, we would get $t_{\text{cell}} = s_{\text{cell}} = n$.

Let $k = \log |\mathbb{F}|$. Kedlaya and Umans [22] provided a data structure for the polynomial evaluation that uses space $n^{1+\varepsilon} \cdot k^{1+o(1)}$ and query time $\log^{O(1)} n \cdot k^{1+o(1)}$. Note that, $n \cdot k$ is the size of the input and k is the size of the output.

The first lower bound for the cell probe model was given by Miltersen [26]. He proved that for any cell probe data structure for the polynomial evaluation it must hold that $t_{\text{cell}} \geq \Omega(k / \log s_{\text{cell}})$. This was improved by Larsen [23] to $t_{\text{cell}} \geq \Omega(k / \log(s_{\text{cell}} w / nk))$, that gives $t_{\text{cell}} \geq \Omega(k)$ if the data structure uses linear space $s_{\text{cell}} \cdot w = O(n \cdot k)$. However, the size of \mathbb{F} has to be super-linear, i.e., $|\mathbb{F}| \geq n^{1+\Omega(1)}$, and it is not known if the bound holds for smaller fields, e.g., of linear size. Data structures in a bit probe model were studied by Gál and Miltersen [15]. The bit probe model is the same as the cell probe model but each cell contains only a single bit, i.e., $w = 1$. They studied succinct data structures that are data structures such that $s_{\text{cell}} = (n + r) \cdot k$ for $r < o(n)$. Thus, the succinct data structures are related to systematic data structures but still, the succinct data structures are charged for any probe (as any other data structure in the cell probe model). Note that a succinct data structure stores only a few more bits than it is needed due to information-theoretic requirement. Gál and Miltersen [15] showed that for any succinct data structure in the bit probe model it holds that $r \cdot t_{\text{cell}} \geq \Omega(n \cdot k)$. We are not aware of any lower bound for systematic data structures for the polynomial evaluation.

Larsen et al. [24] also gave a log-squared lower bound for dynamic data structures in the cell probe model. Dynamic data structures also support updates of the polynomial p which usually impacts their query time.

There is a great interest in algorithmic questions about the polynomial interpolation such as how fast we can interpolate polynomials [16, 5, 18], how many probes we need to interpolate a polynomial if it is given by oracle [6, 20], how to compute the interpolation in a numerically stable way over infinite fields [29] and many others. However, we are not aware of any results about data structures for the interpolation, i.e., when the interpolation algorithm has an access to some precomputed advice.

3 Network Coding

We prove our conditional lower bounds based on the Network Coding Conjecture. In network coding, we are interested in how much information we can send through a given network. A *network* consists of a graph $G = (V, E)$, positive capacities of edges $c : E \rightarrow \mathbb{R}^+$ and k pairs of vertices $(s_0, t_0), \dots, (s_{k-1}, t_{k-1})$. We say a network $R = (G, c, (s_i, t_i)_{i \in [k]})$ is *undirected* or *directed (acyclic)* if the graph G is undirected or directed (acyclic). We say a network is *uniform* if the capacities of all edges in the network equal to some $q \in \mathbb{R}^+$ and we denote such network as $(G, q, (s_i, t_i)_{i \in [k]})$.

A goal of a coding scheme for directed acyclic network $R = (G, c, (s_i, t_i)_{i \in [k]})$ is that at each target t_i it will be possible to reconstruct an input message w_i which was generated at the source s_i . The coding scheme specifies messages sent from each vertex along the outgoing edges as a function of received messages. Moreover, the lengths of the messages sent along the edges have to respect the edge capacities.

More formally, each source s_i of a network receives an input message w_i sampled (independently of the messages for the other sources) from the uniform distribution \mathbf{W}_i on a set W_i . Without loss of generality we can assume that each source s_i has an in-degree 0 (otherwise we can add a vertex s'_i and an edge (s'_i, s_i) and replace s_i by s'_i). There is an alphabet Σ_e for each edge $e \in E(G)$. For each source s_i and each outgoing edge $e = (s_i, u)$ there is a function $f_{s_i, e} : W_i \rightarrow \Sigma_e$ which specifies the message sent along the edge e as a

function of the received input message $w_i \in W_i$. For each non-source vertex $v \in V, v \neq s_i$ and each outgoing edge $e = (v, u)$ there is a similar function $f_{v,e} : \prod_{e'=(u',v)} \Sigma_{e'} \rightarrow \Sigma_e$ which specifies the message sent along the edge e as a function of the messages sent to v along the edges incoming to v . Finally, each target t_i has a decoding function $d_i : \prod_{e'=(u',t_i)} \Sigma_{e'} \rightarrow W_i$. The coding scheme is executed as follows:

1. Each source s_i receives an input message $w_i \in W_i$. Along each edge $e = (s_i, u)$ a message $f_{s_i,e}(w_i)$ is sent.
2. When a vertex v receives all messages m_1, \dots, m_a along all incoming edges (u', v) it sends along each outgoing edge $e = (v, u)$ a message $f_{v,e}(m_1, \dots, m_a)$. As the graph G is acyclic, this procedure is well-defined and each vertex of non-zero out-degree will eventually send its messages along its outgoing edges.
3. At the end, each target t_i computes a string $\tilde{w}_i = d_i(m'_1, \dots, m'_b)$ where m'_j denotes the received messages along the incoming edges (u', t_i) . We say the encoding scheme is *correct* if $\tilde{w}_i = w_i$ for all $i \in [k]$ and any input messages $w_0, \dots, w_{k-1} \in W_0 \times \dots \times W_{k-1}$.

The coding scheme has to respect the edge capacities, i.e., if \mathbf{M}_e is a random variable that represents a message sent along the edge e , then $H(\mathbf{M}_e) \leq c(e)$, where $H(\cdot)$ denotes the Shannon entropy. A *coding rate* of a network R is the maximum r such that there is a correct coding scheme for input random variables $\mathbf{W}_0, \dots, \mathbf{W}_{k-1}$ where $H(\mathbf{W}_i) = \log |W_i| \geq r$ for all $i \in [k]$. A network coding can be defined also for directed cyclic networks or undirected networks but we will not use it here.

Network coding is related to multicommodity flows. A multicommodity flow for an undirected network $\bar{R} = (\bar{G}, c, (s_i, t_i)_{i \in [k]})$ specifies flows for each commodity i such that they transport as many units of commodity from s_i to t_i as possible. A flow of the commodity i is specified by a function $f^i : V \times V \rightarrow \mathbb{R}^+$ which describes for each pair of vertices (u, v) how many units of the commodity i are sent from u to v . Each function f^i has to satisfy:

1. If u, v are not connected by an edge, then $f^i(u, v) = f^i(v, u) = 0$.
2. For each edge $\{u, v\} \in E(\bar{G})$, it holds that $f^i(u, v) = 0$ or $f^i(v, u) = 0$.
3. For each vertex v that is not the source s_i or the target t_i , it holds that what comes to the vertex v goes out from the vertex v , i.e.,

$$\sum_{u \in V} f^i(u, v) = \sum_{u \in V} f^i(v, u).$$

4. What is sent from the source s_i arrives to the target t_i , i.e.,

$$\sum_{u \in V} f^i(s_i, u) - f^i(u, s_i) = \sum_{u \in V} f^i(u, t_i) - f^i(t_i, u).$$

Moreover, all flows together have to respect the capacities, i.e., for each edge $e = \{u, v\} \in E(\bar{G})$ it must hold that $\sum_{i \in [k]} f^i(u, v) + f^i(v, u) \leq c(e)$. A *flow rate* of a network \bar{R} is the maximum r such that there is a multicommodity flow $F = (f^0, \dots, f^{k-1})$ that for each i transports at least r units of the commodity i from s_i to t_i , i.e., for all i , it holds that $\sum_{u \in V} f^i(u, t_i) - f^i(t_i, u) \geq r$. A multicommodity flow for directed graphs is defined similarly, however, the flows can transport the commodities only in the direction of edges.

Let R be a directed acyclic network of a flow rate r' . It is clear that for a coding rate r of R it holds that $r \geq r'$. As we can send the messages without coding and thus reduce the encoding problem to the flow problem. The opposite inequality does not hold: There is a directed network $R = (G, c, (s_i, t_i)_{i \in [k]})$ such that its coding rate is $\Omega(|V(G)|)$ -times larger than its flow rate as shown by Adler et al. [2]. Thus, the network coding for directed networks provides an advantage over the simple solution given by the maximum flow. However, such a

result is not known for undirected networks. Li and Li [25] conjectured that the network coding does not provide any advantage for undirected networks, thus for any undirected network \bar{R} , the coding rate of \bar{R} equals to the flow rate of \bar{R} . This conjecture is known as *Network Coding Conjecture* (NCC) and we state a weaker version of it below.

For a directed graph $G = (V, E)$ we denote by $\text{un}(G)$ the undirected graph (V, \bar{E}) obtained from G by making each directed edge in E undirected (i.e., replacing each $(u, v) \in E(G)$ by $\{u, v\}$). For a directed acyclic network $R = (G, c, (s_i, t_i)_{i \in [k]})$ we define the undirected network $\text{un}(R) = (\text{un}(G), \bar{c}, (s_i, t_i)_{i \in [k]})$ by keeping the source-target pairs and capacities the same, i.e., $c((u, v)) = \bar{c}(\{u, v\})$.

► **Conjecture 1 (Weaker NCC).** *Let R be a directed acyclic network, r be a coding rate of R and \bar{r} be a flow rate of $\text{un}(R)$. Then, $r = \bar{r}$.*

This conjecture was used to prove a conditional lower bound for sorting algorithms with an external memory [13] and for circuits multiplying two numbers [3].

4 NCC Implies Data Structure Lower Bounds

In this paper, we provide several connections between lower bounds for data structures and other computational models. The first connection is that the Network Coding Conjecture (Conjecture 1) implies lower bounds for data structures for the permutation inversion and the polynomial evaluation and interpolation. Assuming NCC, we show that a query time t of a non-adaptive systematic data structure for any of the above problems satisfies $t \geq \Omega(\log n / \log \log n)$, even if it uses linear space, i.e., the advice string \mathbf{a} has size $\varepsilon n \log n$ for sufficiently small constant $\varepsilon > 0$. Formally, we define $t_{\text{Inv}}(s)$ as a query time of the optimal non-adaptive systematic data structure for the permutation inversion using space at most s . Similarly, we define $t_{\text{Eval}}^{\mathbb{F}}(s)$ and $t_{\text{Interp}}^{\mathbb{F}}(s)$ for the polynomial evaluation and interpolation over \mathbb{F} .

► **Theorem 2.** *Let $\varepsilon > 0$ be a sufficiently small constant. Assuming NCC, it holds that $t_{\text{Inv}}(\varepsilon n \log n) \geq \Omega(\log n / \log \log n)$.*

► **Theorem 3.** *Let \mathbb{F} be a field and n be a divisor of $|\mathbb{F}| - 1$. Let $s = \varepsilon n \log |\mathbb{F}|$ for a sufficiently small constant $\varepsilon > 0$. Then assuming NCC, it holds that $t_{\text{Eval}}^{\mathbb{F}}(s), t_{\text{Interp}}^{\mathbb{F}}(s) \geq \Omega(\log n / \log \log n)$.*

Note that by Theorem 2, assuming NCC, it holds that $s \cdot t \geq \Omega(n \log^2 n / \log \log n)$ for $s = \varepsilon n \log n$ and $t = t_{\text{Inv}}(s)$. The same holds for $t_{\text{Eval}}^{\mathbb{F}}$ and $t_{\text{Interp}}^{\mathbb{F}}$ by Theorem 3. Thus, these conditional lower bounds cross the barrier $\Omega(n \log n)$ for $s \cdot t$ given by the best unconditional lower bounds known for the function inversion [36, 10, 1, 17, 30, 11, 8] and the lower bound for the succinct data structures for the polynomial evaluation by Gál and Miltersen [15]. Note that our lower bound does not contradict Hellman's attack [19] for the permutation inversion, as his data structure is heavily adaptive.

Our lower bound for the data structure for the polynomial evaluation and interpolation is applicable even for linear size field (i.e., linear number of queries). Larsen's lower bound for the data structure for the polynomial evaluation [23] is applicable only for superlinear fields, i.e., $|\mathbb{F}| \geq n^{1+\Omega(1)}$. We give the result for the polynomial evaluation here as it has an analogous proof as the lower bound for the polynomial interpolation and it might illustrate a more general phenomenon that our technique might be applicable to a broader class of functions that contains an involution as a subproblem.

To prove Theorems 2 and 3, we build on the technical machinery of Farhadi et al. [13]. The proof can be divided into two steps:

1. From a data structure for the problem we derive a network R with $O(tn)$ edges such that R admits an encoding scheme that is correct on a large fraction of the inputs. This step is distinct for each problem and the reduction for the function inversion is shown in Section 5, the reduction for the polynomial problems is left to the full version [12]. This step uses new ideas and interestingly, it uses the data structure twice in a sequence.
2. If there is a network R with dn edges that admits an encoding scheme which is correct for a large fraction of inputs, then $d \geq \Omega(\log n / \log \log n)$. This step is common to all the problems. It was implicitly proved by Farhadi et al. [13] and Afshani et al. [3].

5 NCC Implies a Weak Lower Bound for the Function Inversion

In this section, we prove Theorem 2 that assuming NCC, any non-adaptive systematic data structure for the permutation inversion requires query time at least $\Omega(\log n / \log \log n)$ even if it uses linear space. Let \mathcal{D} be a data structure for inverting permutations of a linear space $s = \varepsilon n \log n$, for sufficiently small constant $\varepsilon < 1$, with query time $t = t_{\text{inv}}(s)$. Recall that $t_{\text{inv}}(s)$ is a query time of the optimal non-adaptive systematic data structure for the permutation inversion using space s . From \mathcal{D} we construct a directed acyclic network $R = (G, c, (s_i, t_i)_{i \in [n]})$ and an encoding scheme of a coding rate $\log n$. By Conjecture 1 we get that the flow rate of $\text{un}(R) = (\bar{G}, c, (s_i, t_i)_{i \in [n]})$ is $\log n$ as well. We prove that there are many source-target pairs of distance at least $\Omega(\log_t n)$. Since the number of edges of \bar{G} will be $O(tn)$ and flow rate of $\text{un}(R)$ is $\log n$, we are able to derive a lower bound $t \geq \Omega(\log n / \log \log n)$.

We will design the network based on the *probe graph* of the data structure. By the probe graph of the data structure, we understand a graph with n input vertices corresponding to possible oracle probes and n output vertices corresponding to possible data structure queries. Each query vertex is connected to the vertices of oracle probes executed by the data structure when answering that query. Here, we use the non-adaptivity of the studied data structures as the probe graph does not depend on the stored data but only on the data structure itself. Our construction will utilize two copies of the probe graph connected in a sequence. The network will have input vertices s_0, \dots, s_{n-1} and output vertices u_0, \dots, u_{n-1} where each target t_i is set to $u_{i+b \bmod n}$ for a suitable constant b . The input vertices s_0, \dots, s_{n-1} correspond to the oracle vertices of the first copy of the probe graph. (See Fig. 1 for an illustration.)

We will feed distinct values $x_0, \dots, x_{n-1} \in [n]$ to the input vertices which then send them to the query vertices of the first copy of the probe graph. Values x_0, \dots, x_{n-1} define a permutation $f(i) = x_i$. Each query vertex j of the first copy of the probe graph can determine $f^{-1}(j)$ if it is provided with the advice string \mathbf{a}_f of the data structure corresponding to f . (We will fix the most common advice string \mathbf{a}_f and restrict ourselves to inputs x_0, \dots, x_{n-1} consistent with it.) Each query vertex j can also determine the value of a newly defined function $h(j) = f^{-1}(j) + b$ which it sends along its outgoing edges. The second copy of the data structure serves to invert the function h similarly to inverting f . The oracle vertices of the second copy of the probe graph coincide with the query vertices of the first copy. The query vertices of the second copy of the probe graph are the output vertices u_0, \dots, u_{n-1} . Hence, the query vertex $i + b$ of the second copy will be used to determine $h^{-1}(i + b) = x_i$. Thus, x_i is directed from s_i to $t_i = u_{i+b}$.

The above construction gives a network R with an encoding scheme E that is correct only on a substantial fraction of all possible inputs. Namely on inputs $x_0, \dots, x_{n-1} \in [n]$ which are distinct and consistent with the fixed advices. This forces correlations among messages

received by the sources. However, the Network Coding Conjecture requires independently sampled messages for each source. To overcome this issue we use the technique introduced by Farhadi et al. [13] to augment R so that it admits an encoding scheme for independent messages. We provide technical details next.

Let $R = (G, c, (s_i, t_i)_{i \in [k]})$ be a directed acyclic network. Let each source receive a binary string of length r as its input message, i.e., each $W_i = \{0, 1\}^r$. If we concatenate all input messages w_i we get a string of length $r \cdot k$, thus the set of all possible inputs for an encoding scheme for R corresponds to the set $\mathcal{I} = \{0, 1\}^{rk}$. We say an encoding scheme is correct on an input $\bar{w} = (w_0, \dots, w_{k-1}) \in \mathcal{I}$ if it is possible to reconstruct all messages w_i at appropriate targets. An (ε, r) -encoding scheme is an encoding scheme which is correct on at least $2^{(1-\varepsilon)rk}$ inputs in \mathcal{I} .

We say a directed network $R = (G, c, (s_i, t_i)_{i \in [k]})$ is (δ, d) -long if for at least δk source-target pairs (s_i, t_i) , it holds that distance between s_i and t_i in $\text{un}(G)$ is at least d . Here, we measure the distance in the undirected graph $\text{un}(G)$, even though the network R is directed. The following lemma is implicitly used by Farhadi et al. [13] and Afshani et al. [3].

► **Lemma 4** (Implicitly used in [13, 3]). *Let $R = (G, r, (s_i, t_i)_{i \in [k]})$ be a (δ, d) -long directed acyclic uniform network for $\delta > \frac{5}{6}$ and sufficiently large $r \in \mathbb{R}^+$. Assume there is an (ε, r) -encoding scheme for R for sufficiently small ε . Then assuming NCC, it holds that $\frac{|E(G)|}{k} \geq \delta' \cdot d$, where $\delta' = \frac{\delta-5/6}{10}$.*

Now we are ready to prove a conditional lower bound for the permutation inversion. For the proof we use the following fact which follows from well-known Stirling's formula:

► **Fact 1.** *The number of permutations $[n] \rightarrow [n]$ is at least $2^{n \log n - 2n}$.*

► **Theorem 2.** *Let $\varepsilon > 0$ be a sufficiently small constant. Assuming NCC, it holds that $t_{\text{inv}}(\varepsilon n \log n) \geq \Omega(\log n / \log \log n)$.*

Proof Sketch. Let $\mathcal{D} = \mathcal{D}_n$ be the optimal data structure for the inversion of permutation on $[n]$ using space $\varepsilon n \log n$. We set $t = t_{\text{inv}}(\varepsilon n \log n)$. We will construct a directed acyclic uniform network $R = (G, r, (s_i, t_i)_{i \in [n]})$ where $r = \log n$. Let $\varepsilon' = 2 \cdot \varepsilon + \frac{2}{q} + \frac{2}{\log n}$ for sufficiently large q so that we could apply Lemma 4. The network R will admit an (ε', r) -encoding scheme E . The number of edges of G will be at most $2tn$ and the network R will be $(\frac{9}{10}, d)$ -long for $d = \frac{1}{2} \log_{qt} n$. Thus, by Lemma 4 we get that

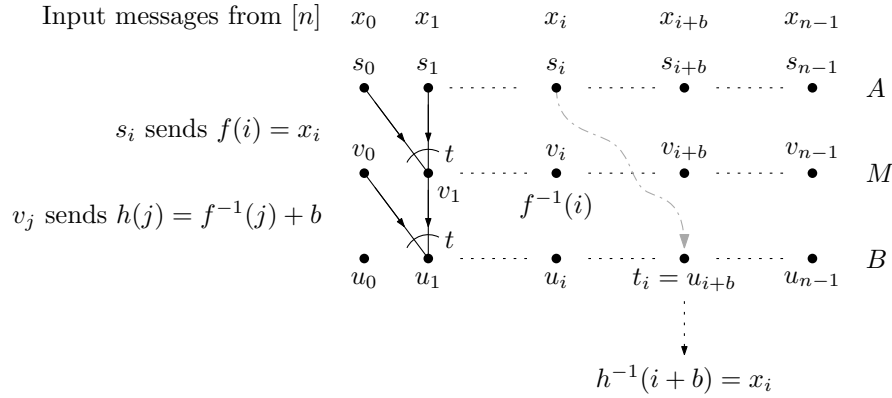
$$2t = \frac{2tn}{n} \geq \Omega(\log_{qt} n),$$

from which we can conclude that $t \geq \Omega(\log n / \log \log n)$. Thus, it remains to construct the network R and the scheme E .

First, we construct a graph G' which will yield the graph G by deleting some edges. The graph G' has three layers of n vertices: a source layer A of n sources s_0, \dots, s_{n-1} , a middle layer M of n vertices v_0, \dots, v_{n-1} and a target layer B of n vertices u_0, \dots, u_{n-1} . The targets t_0, \dots, t_{n-1} of R will be assigned to the vertices u_0, \dots, u_{n-1} later.

We add edges according to the data structure \mathcal{D} : Let $Q_j \subseteq [n]$ be the set of oracle probes which \mathcal{D} makes during the computation of $f^{-1}(j)$, i.e., for each $i \in Q_j$, the query j probes the oracle f for $f(i)$. As \mathcal{D} is non-adaptive, the sets Q_j are well-defined. For each $j \in [n]$ and $i \in Q_j$ we add edges (s_i, v_j) and (v_i, u_j) . We set a capacity of all edges to $r = \log n$. This finishes the construction of G' , see Fig. 1 for illustration of the graph G' .

The graph G' has exactly $2tn$ edges. Moreover, the vertices of the middle and the target layer have in-degree at most t as the incoming edges correspond to the oracle probes made by \mathcal{D} . However, some vertices of the source and the middle layer might have large outdegree,



■ **Figure 1** A sketch of the graph G' and encoding scheme E .

which is a problem that might prevent the network R to be $(\frac{9}{10}, d)$ -long. Thus, we need to remove edges adjacent to high-degree vertices. Let $W \subseteq V(G')$ be the set of vertices of out-degree larger than qt . We remove all edges incident to W from G' to obtain the graph G . (For simplicity, we keep the degree 0 vertices in G). Thus, the maximum degree of G is at most qt . Since the graph G' has $2tn$ edges, it holds that $|W| \leq \frac{2}{q} \cdot n$.

Now, we assign the targets of R in such a way that R is $(\frac{9}{10}, d)$ -long. Let C_v be the set of vertices of G which have distance at most d from v in $\text{un}(G)$. Since the maximum degree of G is at most qt and $d = \frac{1}{2} \log_{qt} n$, for each $v \in V(G)$, $|C_v| \leq 2\sqrt{n}$. It follows from an averaging argument that there is an integer b such that there are at least $n - 2\sqrt{n}$ sources s_i with distance at least d from u_{i+b} in $\text{un}(G)$. (Here the addition $i + b$ is modulo n .) We fix one such b and set $t_i = u_{i+b}$. For n large enough, it holds that $n - 2\sqrt{n} \geq \frac{9}{10} \cdot n$. Thus, the network R is $(\frac{9}{10}, d)$ -long.

It remains to construct the (ε', r) -encoding scheme E for R (see Fig. 1 for a sketch of the encoding E). We only sketch the construction here, the full proof is in the full version [12]. At vertices of the middle layer we compute the inversion of f , i.e., at a vertex v_j we compute $f^{-1}(j)$ using the data structure \mathcal{D} . To do that we need to fix the advice string \mathbf{a}_f (of at most $\varepsilon n \log n$ bits) and values on the vertices in W (the high degree vertices).

We define a function $h : [n] \rightarrow [n]$ as $h(j) = f^{-1}(j) + b$. Thus, at each vertex v_j we are able to compute the value $h(j)$. By the analogous strategy, we compute the inverse $h^{-1}(\ell)$ at each vertex u_ℓ (again using the data structure \mathcal{D} , now for the function h). It can be showed that $h^{-1}(i + b) = x_i$, thus we can reconstruct x_i at each vertex $t_i = u_{i+b}$. Overall, we fix at most $(2\varepsilon + \frac{2}{q}) \cdot n \log n$ bits (the advice strings \mathbf{a}_f and \mathbf{a}_h , and the computed values at the vertices in W). Since the scheme E is correct on all inputs encoding a permutation and consistent with the fixing, the scheme E is (ε', r) -encoding scheme for $\varepsilon' = 2\varepsilon + \frac{2}{q} + \frac{2}{\log n}$. ◀

6 Strong Lower Bounds for Data Structures and Lower Bounds for Boolean Circuits

In this section, we study a connection between non-adaptive data structures and boolean circuits. We are interested in circuits with binary AND and OR gates, and unary NOT gates. (See e.g. [21] for background on circuits). Corrigan-Gibbs and Kogan [9] describe a connection between lower bounds for non-adaptive data structures and lower bounds for boolean circuits for a special case when the data structure computes function inversion. They

show that we would get a circuit lower bound if any non-adaptive data structure using $O(N^\varepsilon)$ queries must use at least $\omega(N/\log \log N)$ bits of advice (for some fixed constant $\varepsilon > 0$). They define a *boolean operator* to be a family of functions $(F_N)_{N \in \mathbb{N}}$ for $F_N: \{0, 1\}^N \rightarrow \{0, 1\}^N$ represented by boolean circuits with N input and N output bits and constant fan-in gates. A boolean operator is said to be an *explicit operator* if the decision problem whether the j -th output bit of F_N is equal to one is in the complexity class NP.

► **Theorem 5** (Corrigan-Gibbs and Kogan [9], Theorem 6 (in contraposition)). *If every explicit operator F_N has fan-in-two boolean circuits of size $O(N)$ and depth $O(\log N)$ then, for every $\varepsilon > 0$, there exists a family of strongly non-adaptive black-box algorithms that inverts all functions $f: [n] \rightarrow [n]$ using $O(n \log n / \log \log n)$ bits of advice and $O(n^\varepsilon)$ online probes.*

To prove their theorem Corrigan-Gibbs and Kogan [9] use the common bits model of boolean circuits introduced by Valiant [31, 32, 33]. Valiant proves that for any circuit there is a small cut, called *common bits*, such that each output bit is connected just to few input bits. Corrigan-Gibbs and Kogan [9] use the common bits of the given circuit to create a non-adaptive data structure by setting the advice string to the content of common bits and the queries are to those function values which are still connected to the particular output after removing the common bits. Using Valiant's result directly one can obtain the following corollaries (see the full version [12] for details).

► **Corollary 6.** *Let $\mathcal{S} = \{p^k \mid p \text{ is a prime}, k \in \mathbb{N}, k \neq 0\}$ be the set of all sizes of finite fields. For each $n \in \mathcal{S}$, let $\mathbb{F}_n = GF(n)$ and σ_n be a primitive $(n-1)$ -th root of unity (thus a generator of the multiplicative group \mathbb{F}_n^*). If there is a circuit family computing $FFFT_{n-1, \sigma_n}$ (over \mathbb{F}_n) of size $O(n \log n)$ and depth $O(\log n)$ (where each input and output number is represented by $\log |\mathbb{F}_n|$ bits) then for every $\varepsilon > 0$ there is a family of non-adaptive data structures $\{\mathcal{D}_n\}_{n \in \mathcal{S}}$ where \mathcal{D}_n uses advice of size $O(n \log n / \log \log n)$ and on a query $j \in [n-1]$ outputs the j -th output of $FFFT_{n-1, \sigma_n}$ using $O(n^\varepsilon)$ queries to the input.*

To put the corollary in a counter-positive way: if for some $\varepsilon > 0$, there are no non-adaptive data structures for polynomial interpolation, polynomial evaluation, or FFFT with an advice of size $o(n \log n / \log \log n)$ that use $O(n^\varepsilon)$ queries to the input then there are no linear-size circuits of logarithmic depth for FFFT.

In Theorem 2, resp. Theorem 3, we prove a conditional lower bound for permutation inversion, resp. polynomial evaluation and polynomial interpolation, of the form, that a non-adaptive data structure using $\varepsilon n \log n$ bits must do at least $\Omega(\log n / \log \log n)$ queries. It is not clear if assuming NCC we can get a sufficiently strong lower bound which would rule out non-adaptive data structures with sublinear advice string using $O(n^\varepsilon)$ oracle queries.

► **Corollary 7.** *We say that a circuit $C_n: \{0, 1\}^{n \lceil \log n \rceil} \rightarrow \{0, 1\}^{n \lceil \log n \rceil}$ sorts its input if on an input viewed as n binary strings $x_1, x_2, \dots, x_n \in \{0, 1\}^{\lceil \log n \rceil}$ outputs the strings sorted lexicographically. If there is a circuit family $(C_n)_{n \in \mathbb{N}}$, where $C_n: \{0, 1\}^{n \lceil \log n \rceil} \rightarrow \{0, 1\}^{n \lceil \log n \rceil}$ sorts its inputs, and each circuit C_n is of size $O(n \log n)$ and depth $O(\log n)$ then for every $\varepsilon > 0$, for every permutation $f: [n] \rightarrow [n]$ there is a non-adaptive data structure for inverting f that uses advice of size $O(n \log n / \log \log n)$ and $O(n^\varepsilon)$ queries.*

The works of Farhadi et al. [13] and Asharov et al. [4] connect the NCC conjecture directly to lower bounds for sorting. Their work studies sorting n numbers of $k + w$ bits by their first k bits. Namely Asharov et al. [4] show that NCC implies that constant fan-in constant fan-out circuits must have size $\Omega(nk(w - \log(n) + k))$ whenever $w > \log(n) - k$ and $k \leq \log n$. This is incomparable to our results as we have $w = 0$.

References

- 1 Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 357–379. Springer, 2017. doi:10.1007/978-3-319-70697-9_13.
- 2 Micah Adler, Nicholas J. A. Harvey, Kamal Jain, Robert Kleinberg, and April Rasala Lehman. On the capacity of information networks. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA '06*, page 241–250, USA, 2006. Society for Industrial and Applied Mathematics.
- 3 Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. Lower Bounds for Multiplication via Network Coding. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:12, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.10.
- 4 Gilad Asharov, Wei-Kai Lin, and Elaine Shi. Sorting short keys in circuits of size $o(n \log n)$. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2249–2268. SIAM, 2021.
- 5 Michael Ben-Or and Prason Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88*, page 301–309, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/62212.62241.
- 6 Michael Clausen, Andreas Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k -sparse multivariate polynomials over finite fields. *Theor. Comput. Sci.*, 84(2):151–164, July 1991. doi:10.1016/0304-3975(91)90157-W.
- 7 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
- 8 Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John Steinberger. Random oracles and non-uniformity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2018 Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 227–258. Springer Verlag, 2018. 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2018 ; Conference date: 29-04-2018 Through 03-05-2018. doi:10.1007/978-3-319-78381-9_9.
- 9 Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography*, pages 393–421, Cham, 2019. Springer International Publishing.
- 10 Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and prgs. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, pages 649–665, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- 11 Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017*, pages 473–495, Cham, 2017. Springer International Publishing.
- 12 Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová. Data structures lower bounds and popular conjectures, 2021. arXiv:2102.09294.
- 13 Alireza Farhadi, MohammadTaghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. In *Proceedings of the 51st*

- Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 997–1008, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316337.
- 14 Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. doi:10.1137/S0097539795280512.
 - 15 Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 332–344, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
 - 16 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, USA, 3rd edition, 2013.
 - 17 Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005. doi:10.1137/S0097539704443276.
 - 18 Dima Grigoryev, Marek Karpinski, and Michael Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, 19:1059–1063, December 1990. doi:10.1137/0219073.
 - 19 M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. doi:10.1109/TIT.1980.1056220.
 - 20 Gábor Ivanyos, Marek Karpinski, Miklos Santha, Nitin Saxena, and Igor E. Shparlinski. Polynomial interpolation and identity testing from high powers over finite fields. *Algorithmica*, 80(2):560–575, 2018. doi:10.1007/s00453-016-0273-1.
 - 21 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.
 - 22 K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–155, 2008. doi:10.1109/FOCS.2008.13.
 - 23 Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS ’12, page 293–301, USA, 2012. IEEE Computer Society. doi:10.1109/FOCS.2012.21.
 - 24 Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 978–989, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188790.
 - 25 Zongpeng Li and Baochun Li. Network coding: The case of multiple unicast sessions. *Proceedings of the 42nd Allerton Annual Conference on Communication, Control, and Computing*, October 2004.
 - 26 Peter Bro Miltersen. On the cell probe complexity of polynomial evaluation. *Theor. Comput. Sci.*, 143(1):167–174, 1995. doi:10.1016/0304-3975(95)80032-5.
 - 27 John M Pollard. The fast fourier transform in a finite field. *Mathematics of computation*, 25(114):365–374, 1971.
 - 28 Jean-Pierre Serre. *A course in arithmetic*, volume 7. Springer Science & Business Media, 2012.
 - 29 A. Smoktunowicz, I. Wróbel, and P. Kosowski. A new efficient algorithm for polynomial interpolation. *Computing*, 79(1):33–52, February 2007. doi:10.1007/s00607-006-0185-z.
 - 30 Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 205–223, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
 - 31 Leslie G Valiant. Graph-theoretic arguments in low-level complexity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 162–176. Springer, 1977.
 - 32 Leslie G Valiant. Exponential lower bounds for restricted monotone circuits. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 110–117, 1983.
 - 33 Leslie G Valiant. Why is boolean complexity theory difficult. *Boolean Function Complexity*, 169(84-94):4, 1992.

- 34 Emanuele Viola. On the power of small-depth computation. *Found. Trends Theor. Comput. Sci.*, 5(1):1–72, 2009.
- 35 Emanuele Viola. Lower bounds for data structures with space close to maximum imply circuit lower bounds. *Theory of Computing*, 15(18):1–9, 2019. doi:10.4086/toc.2019.v015a018.
- 36 A. C.-C. Yao. Coherent functions and program checkers. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 84–94, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/100216.100226.

Approximation Schemes for Bounded Distance Problems on Fractionally Treewidth-Fragile Graphs

Zdeněk Dvořák 

Charles University, Prague, Czech Republic

Abhiruk Lahiri 

Ariel University, Israel

Abstract

We give polynomial-time approximation schemes for monotone maximization problems expressible in terms of distances (up to a fixed upper bound) and efficiently solvable on graphs of bounded treewidth. These schemes apply in all fractionally treewidth-fragile graph classes, a property which is true for many natural graph classes with sublinear separators. We also provide quasipolynomial-time approximation schemes for these problems in all classes with sublinear separators.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation, sublinear separators

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.40

Related Version *Full Version:* <https://arxiv.org/abs/2105.01780>

Funding *Zdeněk Dvořák:* Supported by the ERC-CZ project LL2005 (Algorithms and complexity within and beyond bounded expansion) of the Ministry of Education of Czech Republic.

Abhiruk Lahiri: Supported by ISF grant 822/18 and Ariel University Post-doctoral fellowship.

Acknowledgements The second author wish to thank the Caesarea Rothschild Institute and the Department of Computer Science, University of Haifa, for providing its facilities for his research activities.

1 Introduction

In this paper, we consider optimization problems such as:

- **MAXIMUM r -INDEPENDENT SET**, $r \in \mathbb{Z}^+$: Given a graph G , the objective is to find a largest subset $X \subseteq V(G)$ such that distance in G between any two vertices in X is at least r .
- **MAXIMUM WEIGHT INDUCED FOREST**: Given a graph G and an assignment $w : V(G) \rightarrow \mathbb{Z}_0^+$ of non-negative weights to vertices, the objective is to find a subset $X \subseteq V(G)$ such that $G[X]$ does not contain a cycle and subject to that, $w(X) := \sum_{v \in X} w(v)$ is maximized.
- **MAXIMUM (F, r) -MATCHING**, for a fixed connected graph F and $r \in \mathbb{Z}^+$: Given a graph G , the objective is to find a largest subset $X \subseteq V(G)$ such that $G[X]$ can be partitioned into vertex-disjoint copies of F such that distance in G between any two vertices belonging to different copies is at least r .

To be precise, to fall into the scope of our work, the problem must satisfy the following conditions:

- It must be a **maximization problem on certain subsets of vertices** of an input graph, possibly with non-negative weights. That is, the problem specifies which subsets of vertices of the input graph are *admissible*, and the goal is to find an admissible subset of largest size or weight.



© Zdeněk Dvořák and Abhiruk Lahiri;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 40; pp. 40:1–40:10

Leibniz International Proceedings in Informatics



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

- The problem must be **defined in terms of distances between the vertices, up to some fixed bound**. That is, there exists a parameter $r \in \mathbb{Z}^+$ such that for any graphs G and G' , sets $X \subseteq V(G)$ and $X' \subseteq V(G')$, and a bijection $f : X \rightarrow X'$, if $\min(r, d_G(u, v)) = \min(r, d_{G'}(f(u), f(v)))$ holds for all $u, v \in X$, then X is admissible in G if and only if X' is admissible in G' .
- The problem must be **monotone** (i.e., all subsets of an admissible set must be admissible), or at least **near-monotone** (as happens for example for MAXIMUM (F, r) -MATCHING) in the following sense: There exists a parameter $c \in \mathbb{Z}^+$ such that for any admissible set A in a graph G , there exists a system $\{R_v \subseteq A : v \in A\}$ of subsets of A such that
 - each vertex belongs to R_v for at most c vertices $v \in A$,
 - $v \in R_v$ for each $v \in A$, and
 - for every $Z \subseteq A$, the subset $A \setminus \bigcup_{v \in Z} R_v$ is admissible in G .
- The problem must be **tractable in graphs of bounded treewidth**, that is, there must exist a function g and a polynomial p such that given any graph G , its tree decomposition of width t , an assignment w of non-negative weights to the vertices of G , and a set $X_0 \subseteq X$, it is possible to find a maximum-weight admissible subset of X_0 in time $g(t)p(|V(G)|)$.

Let us call such problems $(\leq r)$ -distance determined c -near-monotone (g, p) -tw-tractable. Note that a convenient way to verify these assumptions is to show that the problem is expressible in *solution-restricted Monadic Second-Order Logic (MSOL) with bounded-distance predicates*, i.e., by a MSOL formula with one free variable X such that the quantification is restricted to subsets and elements of X , and using binary predicates d_1, \dots, d_r , where $d_i(u, v)$ is interpreted as testing whether the distance between u and v in the whole graph is at most i . For example, r -INDEPENDENT SET is expressed by the formula $(\forall u, v \in X) u = v \vee \neg d_r(u, v)$. This ensures that the problem is $(\leq r)$ -distance determined, and $(g, O(n))$ -tw-tractable for some function g by Courcelle's meta-algorithmic result [5].

Of course, the problems satisfying the assumptions outlined above are typically hard to solve optimally, even in rather restrictive circumstances. For example, MAXIMUM INDEPENDENT SET is NP-hard even in planar graphs of maximum degree at most 3 and arbitrarily large (fixed) girth [1]. Moreover, it is hard to approximate it within factor of 0.995 in graphs of maximum degree at most three [4]. Hence, to obtain polynomial-time approximation schemes (PTASes), i.e., polynomial-time algorithms for approximating within any fixed precision, a restriction other than just bounding the maximum degree is needed.

A natural restriction that has been considered in this context is the requirement that the graphs have sublinear separators (a set S of vertices of a graph G is a *balanced separator* if every component of $G \setminus S$ has at most $|V(G)|/2$ vertices, and a hereditary class \mathcal{G} of graphs has *sublinear separators* if for some $c < 1$, every graph $G \in \mathcal{G}$ has a balanced separator of size $O(|V(G)|^c)$). This restriction still lets us speak about many interesting graph classes (planar graphs [19] and more generally proper minor-closed classes [2], many geometric graph classes [21], ...). Moreover, the problems discussed above admit PTASes in all classes with sublinear separators or at least in substantial subclasses of these graphs:

- MAXIMUM INDEPENDENT SET has been shown to admit a PTAS in graphs with sublinear separators already in the foundational paper of Lipton and Tarjan [20].
- For any positive integer, MAXIMUM r -INDEPENDENT SET and several other problems are known to admit PTASes in graphs with sublinear separators by a straightforward local search algorithm [16].
- All of the problems mentioned above (and more) are known to admit PTASes in planar graphs by a layering argument of Baker [3]; this approach can be extended to some related graph classes, including all proper minor-closed classes [6, 12].

- The problems also admit PTASes in graph classes that admit thin systems of overlays [11], a technical property satisfied by all proper minor-closed classes and by all hereditary classes with sublinear separators and bounded maximum degree.
- Bidimensionality arguments [7] apply to a wide range of problems in proper minor-closed graph classes.

However, each of the outlined approaches has drawbacks. On one side, the local search approach only applies to specific problems and does not work at all in the weighted setting. On the other side of the spectrum, Baker's approach is quite general as far as the problems go, but there are many hereditary graph classes with sublinear separators to which it does not seem to apply. The approach through thin systems of overlays tries to balance these concerns, but it is rather technical and establishing this property is difficult.

Another option that has been explored is via *fractional treewidth-fragility*. For a function $f: \mathbb{Z}^+ \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ and a polynomial p , a class of graphs \mathcal{G} is *p-efficiently fractionally treewidth-f-fragile* if there exists an algorithm that for every $k \in \mathbb{Z}^+$ and a graph $G \in \mathcal{G}$ returns in time $p(|V(G)|)$ a collection of subsets $X_1, X_2, \dots, X_m \subseteq V(G)$ such that each vertex of G belongs to at most m/k of the subsets, and moreover, for $i = 1, \dots, m$, the algorithm also returns a tree decomposition of $G \setminus X_i$ of width at most $f(k, |V(G)|)$. We say a class is *p-efficiently fractionally treewidth-fragile* if f does not depend on its second argument (the number of vertices of G). This property turns out to hold for basically all known natural graph classes with sublinear separators. In particular, a hereditary class \mathcal{G} of graphs is efficiently fractionally treewidth-fragile if

- \mathcal{G} has sublinear separators and bounded maximum degree [9],
- \mathcal{G} is proper minor-closed [8, 12], or
- \mathcal{G} consists of intersection graphs of convex objects with bounded aspect ratio in a finite-dimensional Euclidean space and the graphs have bounded clique number, as can be seen by a modification of the argument of Erlebach et al. [15]. This includes all graph classes with polynomial growth [18].

In fact, Dvořák conjectured that every hereditary class with sublinear separators is fractionally treewidth-fragile, and gave the following result towards this conjecture.

► **Theorem 1** (Dvořák [10]). *There exists a polynomial p so that the following claim holds. For every hereditary class \mathcal{G} of graphs with sublinear separators, there exists a polynomial q such that \mathcal{G} is p -efficiently fractionally treewidth-f-fragile for the function $f(k, n) = q(k \log n)$.*

Moreover, Dvořák [9] observed that weighted MAXIMUM INDEPENDENT SET admits a PTAS in any efficiently fractionally treewidth-fragile class of graphs. Indeed, the algorithm is quite simple, based on the observation that for the sets X_1, \dots, X_m from the definition of fractional treewidth-fragility, at least one of the graphs $G \setminus X_1, \dots, G \setminus X_m$ (of bounded treewidth) contains an independent set whose weight is within the factor of $1 - 1/k$ from the optimal solution. A problem with this approach is that it does not extend to more general problems; even for the MAXIMUM 2-INDEPENDENT SET problem, the approach fails, since a 2-independent set in $G \setminus X_i$ is not necessarily 2-independent in G . Indeed, this observation served as one of the motivations behind more restrictive (and more technical) concepts employed in [11, 12].

As our main result, we show that this intuition is in fact false: There is a simple way how to extend the approach outlined in the previous paragraph to all bounded distance determined near-monotone tw-tractable problems.

► **Theorem 2.** *For every class \mathcal{G} of graphs with bounded expansion, there exists a function $h : \mathbb{Z}^+ \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ such that the following claim holds. Let c and r be positive integers, $g : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ and $f : \mathbb{Z}^+ \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ functions and p and q polynomials. If \mathcal{G} is q -efficiently fractionally treewidth- f -fragile, then for every $(\leq r)$ -distance determined c -near-monotone (g, p) -tw-tractable problem, there exists an algorithm that given a graph $G \in \mathcal{G}$, an assignment of non-negative weights to vertices, and a positive integer k , returns in time $h(r, c)|V(G)| + q(|V(G)|) \cdot p(|V(G)|) \cdot g(f(h(r, c)k, |V(G)|))$ an admissible subset of $V(G)$ whose weight is within the factor of $1 - 1/k$ from the optimal one.*

Note that the assumption that \mathcal{G} has bounded expansion is of little consequence – it is satisfied for any hereditary class with sublinear separators [14] as well as for any fractionally treewidth-fragile class [9]; see Section 2 for more details. The time complexity of the algorithm from Theorem 2 is polynomial if f does not depend on its second argument, and quasipolynomial (exponential in a polylogarithmic function) if f is logarithmic in the second argument and g is single-exponential, i.e., if $g(n) = \exp(n^{O(1)})$. Hence, we obtain the following corollaries.

► **Corollary 3.** *Let c and r be positive integers, $g : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ a function and p a polynomial. Every $(\leq r)$ -distance determined c -near-monotone (g, p) -tw-tractable problem admits a PTAS in any efficiently fractionally treewidth-fragile class of graphs.*

We say a problem admits a quasipolynomial-time approximation schemes (QPTAS) if there exist quasipolynomial-time algorithms for approximating the problem within any fixed precision. Combining Theorems 1 and 2, we obtain the following result.

► **Corollary 4.** *Let c and r be positive integers, $g : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ a single-exponential function, and p a polynomial. Every $(\leq r)$ -distance determined c -near-monotone (g, p) -tw-tractable problem admits a QPTAS in any hereditary class of graphs with sublinear separators.*

The idea of the algorithm from Theorem 2 is quite simple: We consider the sets X_1, \dots, X_m from the definition of fractional treewidth- f -fragility, extend them to suitable supersets Y_1, \dots, Y_m , and argue that for $i = 1, \dots, m$, any admissible set in $G \setminus X_i$ disjoint from Y_i is also admissible in G , and that for some i , the weight of the heaviest admissible set in $G \setminus X_i$ disjoint from Y_i is within the factor of $1 - 1/k$ from the optimal one. The construction of the sets Y_1, \dots, Y_m is based on the existence of orientations with bounded outdegrees that represent all short paths, a generalization of a result Kowalik and Kurowski [17] that we present in Section 2.

Let us remark one can develop the idea of this paper in further directions. Dvořák proved in [13] (via a substantially more involved argument) that every monotone maximization problem expressible in first-order logic admits a PTAS in any efficiently fractionally treewidth-fragile class of graphs. Note that this class of problems is incomparable with the one considered in this paper (e.g., MAXIMUM INDUCED FOREST is not expressible in first-order logic, while MAXIMUM INDEPENDENT SET consisting of vertices belonging to triangles is expressible in first-order logic but does not fall into the scope of the current paper).

Finally, it is worth mentioning that our results only apply to maximization problems. We were able to extend the previous uses of fractional treewidth-fragility by giving a way to handle dependencies over any bounded distance. However, for the minimization problems, we do not know whether fractional treewidth-fragility is sufficient even for the distance-1 problems. For a simple example, consider the MINIMUM VERTEX COVER problem in fractionally treewidth-fragile graphs, or more generally in hereditary classes with sublinear separators. While the unweighted version can be dealt with by the local search method [16], we do not know whether there exists a PTAS for the weighted version of this problem.

2 Paths and orientations in graphs with bounded expansion

For $r \in \mathbb{Z}_0^+$, a graph H is an r -shallow minor of a graph G if H can be obtained from a subgraph of G by contracting pairwise vertex-disjoint connected subgraphs, each of radius at most r . For a function $f: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$, a class \mathcal{G} of graphs has *expansion bounded* by f if for all non-negative integers r , all r -shallow minors of graphs from \mathcal{G} have average degree at most $f(r)$. A class has *bounded expansion* if its expansion is bounded by some function f . The theory of graph classes with bounded expansion has been developed in the last 15 years, and the concept has found many algorithmic and structural applications; see [23] for an overview. Crucially for us, this theory includes a number of tools for dealing with short paths. Moreover, as we have pointed out before, all hereditary graph classes with sublinear separators [14] as well as all fractionally treewidth-fragile classes [9] have bounded expansion.

Let \vec{G} be an orientation of a graph G , i.e., uv is an edge of G if and only if the directed graph \vec{G} contains at least one of the directed edges (u, v) and (v, u) ; note that we allow \vec{G} to contain both of them at the same time, and thus for the edge uv to be oriented in both directions. We say that a directed graph \vec{H} with the same vertex set is a *1-step fraternal augmentation* of \vec{G} if $\vec{G} \subseteq \vec{H}$, for all distinct edges $(x, y), (x, z) \in E(\vec{G})$, either (y, z) or (z, y) is an edge of \vec{H} , and for each edge $(y, z) \in E(\vec{H}) \setminus E(\vec{G})$, there exists a vertex $x \in V(\vec{G}) \setminus \{y, z\}$ such that $(x, y), (x, z) \in E(\vec{G})$. That is, to obtain \vec{H} from \vec{G} , for each pair of edges $(x, y), (x, z) \in E(\vec{G})$ we add an edge between y and z in one of the two possible directions (we do not specify the direction, but in practice we would choose directions of the added edges that minimize the maximum outdegree of the resulting directed graph). For an integer $a \geq 0$, we say \vec{F} is an *a-step fraternal augmentation* of \vec{G} if there exists a sequence $\vec{G} = \vec{G}_0, \vec{G}_1, \dots, \vec{G}_a = \vec{F}$ where for $i = 1, \dots, a$, \vec{G}_i is a 1-step fraternal augmentation of \vec{G}_{i-1} . We say \vec{F} is an *a-step fraternal augmentation* of an undirected graph G if \vec{F} is an *a-step fraternal augmentation* of some orientation of G .

A key property of graph classes with bounded expansion is the existence of fraternal augmentations with bounded outdegrees. Let us remark that whenever we speak about an algorithm returning an *a-step fraternal augmentation* \vec{H} or taking one as an input, this implicitly includes outputting or taking as an input the whole sequence of 1-step fraternal augmentations ending in \vec{H} .

► **Lemma 5** (Nešetřil and Ossona de Mendez [22]). *For every class \mathcal{G} with bounded expansion, there exists a function $d: \mathbb{Z}_0^+ \rightarrow \mathbb{Z}^+$ such that for each $G \in \mathcal{G}$ and each non-negative integer a , the graph G has an *a-step fraternal augmentation* of maximum outdegree at most $d(a)$. Moreover, such an augmentation can be found in time $O(d(a)|V(G)|)$.*

As shown already in [22], fraternal augmentations can be used to succinctly represent distances between vertices of the graph. For the purposes of this paper, we need a more explicit representation by an orientation of the original graph without the additional augmentation edges, as we only assume that the original (rather than the augmented) graph is fractionally treewidth-fragile. Let us remark that the existence of such a representation was shown by Kowalik and Kurowski [17] in a more restrictive setting of graph classes closed under topological minors.

By a *walk* in a directed graph \vec{G} , we mean a sequence $W = v_0 v_1 v_2 \dots v_b$ such that for $i = 1, \dots, b$, $(v_{i-1}, v_i) \in E(\vec{G})$ or $(v_i, v_{i-1}) \in E(\vec{G})$; that is, the walk does not have to respect the orientation of the edges. The walk W is *inward directed* if for some $c \in \{0, \dots, b\}$, we have $(v_i, v_{i+1}) \in E(\vec{G})$ for $i = 0, \dots, c-1$ and $(v_i, v_{i-1}) \in E(\vec{G})$ for $i = c+1, \dots, b$. For a positive integer r , an orientation \vec{G} of a graph G *represents* ($\leq r$)-distances if for each $u, v \in V(G)$ and each $b \in \{0, \dots, r\}$, the distance between u and v in G is at most b if and

only if \vec{G} contains an inward-directed walk of length at most b between u and v . Note that given such an orientation with bounded maximum outdegree for a fixed r , we can determine the distance between u and v (up to distance r) by enumerating all (constantly many) walks of length at most r directed away from u and away from v and inspecting their intersections.

Our goal now is to show that graphs from classes with bounded expansion admit orientations with bounded maximum outdegree that represent $(\leq r)$ -distances. Let us define a more general notion used in the proof of this claim, adding to the fraternal augmentations the information about the lengths of the walks in the original graph represented by the added edges. A *directed graph with $(\leq r)$ -length sets* is a pair (\vec{H}, ℓ) , where \vec{H} is a directed graph and ℓ is a function assigning a subset of $\{1, \dots, r\}$ to each *unordered* pair $\{u, v\}$ of vertices of \vec{H} , such that if neither (u, v) nor (v, u) is an edge of \vec{H} , then $\ell(\{u, v\}) = \emptyset$. We say that (\vec{H}, ℓ) is an *orientation* of a graph G if G is the underlying undirected graph of \vec{H} and $\ell(\{u, v\}) = \{1\}$ for each $uv \in E(G)$. We say that (\vec{H}, ℓ) is an $(\leq r)$ -*augmentation* of G if $V(\vec{H}) = V(G)$, for each $uv \in E(G)$ we have $1 \in \ell(\{u, v\})$, and for each $u, v \in V(G)$ and $b \in \ell(\{u, v\})$ there exists a walk of length b from u to v in G . Let (\vec{H}_1, ℓ_1) be another directed graph with $(\leq r)$ -length sets. We say (\vec{H}_1, ℓ_1) is a *1-step fraternal augmentation* of (\vec{H}, ℓ) if \vec{H}_1 is a 1-step fraternal augmentation of \vec{H} and for all distinct $u, v \in V(\vec{H})$ and $b \in \{1, \dots, r\}$, we have $b \in \ell_1(\{u, v\})$ if and only if $b \in \ell(\{u, v\})$ or there exist $x \in V(\vec{H}) \setminus \{u, v\}$, $b_1 \in \ell(\{x, u\})$, and $b_2 \in \ell(\{x, v\})$ such that $(x, u), (x, v) \in E(\vec{H})$ and $b = b_1 + b_2$. Note that a 1-step fraternal augmentation of an $(\leq r)$ -augmentation of a graph G is again an $(\leq r)$ -augmentation of G . The notion of an a -step fraternal augmentation of a graph G is then defined in the natural way, by starting with an orientation of G and performing the 1-step fraternal augmentation operation a times. Let us now restate Lemma 5 in these terms (we just need to maintain the edge length sets, which can be done with $O(a^2)$ overhead per operation).

► **Lemma 6.** *Let \mathcal{G} be a class of graphs with bounded expansion, and let $d : \mathbb{Z}_0^+ \rightarrow \mathbb{Z}^+$ be the function from Lemma 5. For each $G \in \mathcal{G}$ and each non-negative integer a , we can in time $O(a^2 d(a) |V(G)|)$ construct a directed graph with $(\leq a + 1)$ -length sets (\vec{H}, ℓ) of maximum outdegree at most $d(a)$ such that (\vec{H}, ℓ) is an a -step fraternal augmentation of G .*

Let (\vec{H}, ℓ) be an $(\leq r)$ -augmentation of a graph G . For $b \leq r$, a *walk of span b* in (\vec{H}, ℓ) is a tuple $(v_0 v_1 \dots v_t, b_1, \dots, b_t)$, where $v_0 v_1 \dots v_t$ is a walk in \vec{H} , $b_i \in \ell(\{v_{i-1}, v_i\})$ for $i = 1, \dots, t$, and $b = b_1 + \dots + b_t$. Note that if there exists a walk of span b from u to v in (\vec{H}, ℓ) , then there also exists a walk of length b from u to v in G . We say that (\vec{H}, ℓ) *represents $(\leq r)$ -distances* in G if for all vertices $u, v \in V(G)$ at distance $b \leq r$ from one another, (\vec{H}, ℓ) contains an inward-directed walk of span b between u and v . Next, we show that this property always holds for sufficient fraternal augmentations.

► **Lemma 7.** *Let G be a graph and r a positive integer and let (\vec{H}, ℓ) be a directed graph with $(\leq r)$ -length sets. If (\vec{H}, ℓ) is obtained as an $(r - 1)$ -step fraternal augmentation of G , then it represents $(\leq r)$ -distances in G .*

Proof. For $b \leq r$, consider any walk $W = (v_0 v_1 \dots v_t, b_1, \dots, b_t)$ of span b in an $(\leq r)$ -augmentation (\vec{H}_1, ℓ_1) of G , and let (\vec{H}_2, ℓ_2) be a 1-step augmentation of (\vec{H}_1, ℓ_1) . Note that W is also a walk of span b between v_0 and v_t in (\vec{H}_2, ℓ_2) . Suppose that W is not inward-directed in (\vec{H}_1, ℓ_1) , and thus there exists $i \in \{1, \dots, t - 1\}$ such that $(v_i, v_{i-1}), (v_i, v_{i+1}) \in E(\vec{H}_1)$. By the definition of 1-step fraternal augmentation, this implies $b_i + b_{i+1} \in \ell_2(\{v_{i-1}, v_{i+1}\})$, and thus $(v_0 \dots v_{i-1} v_{i+1} \dots v_t, b_1, \dots, b_i + b_{i+1}, \dots, b_t)$ is a walk of span b from v_0 to v_t in (\vec{H}_2, ℓ_2) .

Let $(\vec{G}_0, \ell_0), \dots, (\vec{G}_{r-1}, \ell_{r-1})$ be a sequence of $(\leq r)$ -augmentations of G , where (\vec{G}, ℓ_0) is an orientation of G , $(\vec{G}_{r-1}, \ell_{r-1}) = (\vec{H}, \ell)$, and for $i = 1, \dots, r - 1$, (\vec{G}_i, ℓ_i) is a 1-step fraternal augmentation of $(\vec{G}_{i-1}, \ell_{i-1})$. Let u and v be any vertices at distance $b \leq r$ in G ,

and let P be a shortest path between them. Then P naturally corresponds to a walk P_0 of span b in (\vec{G}_0, ℓ_0) . For $i = 1, \dots, r-1$, if P_{i-1} is inward-directed, then let $P_i = P_{i-1}$, otherwise let P_i be a walk of span b in (\vec{G}_i, ℓ_i) obtained from P_{i-1} as described in the previous paragraph. Since each application of the operation decreases the number of vertices of the walk, we conclude that P_{r-1} is an inward-directed walk of span b between u and v in (\vec{H}, ℓ) . Hence, (\vec{H}, ℓ) represents $(\leq r)$ -distances in G . ◀

Next, let us propagate this property back through the fraternal augmentations by orienting some of the edges in both directions. We say that (\vec{H}, ℓ) is an a -step fraternal supraugmentation of a graph G if there exists an a -step fraternal augmentation (\vec{F}, ℓ) of G such that $V(\vec{F}) = V(\vec{H})$, $E(\vec{F}) \subseteq E(\vec{H})$ and for each $(u, v) \in E(\vec{H}) \setminus E(\vec{F})$, we have $(v, u) \in E(\vec{F})$. We say that (\vec{F}, ℓ) is a support of (\vec{H}, ℓ) .

► **Lemma 8.** *Let G be a graph and r a positive integer and let (\vec{H}, ℓ) be an $(\leq r)$ -augmentation of G of maximum outdegree Δ representing $(\leq r)$ -distances. For $a \geq 1$, suppose that (\vec{H}, ℓ) is an a -step fraternal supraugmentation of G . Then we can in time $O(r^2 \Delta |V(G)|)$ obtain an $(a-1)$ -step fraternal supraugmentation of G representing $(\leq r)$ -distances, of maximum outdegree at most $(r+1)\Delta$.*

Proof. Let (\vec{F}, ℓ) be an a -step fraternal augmentation of G forming a support of (\vec{H}, ℓ) , obtained as a 1-step fraternal augmentation of an $(a-1)$ -step fraternal augmentation (\vec{F}_1, ℓ_1) of G . Let (\vec{H}_1, ℓ_1) be the $(a-1)$ -step fraternal supraugmentation of G obtained from (\vec{F}_1, ℓ_1) as follows:

- For all distinct vertices $y, z \in V(G)$ such that $(y, z), (z, y) \in E(\vec{H})$, $(y, z) \in E(\vec{F}_1)$, and $(z, y) \notin E(\vec{F}_1)$, we add the edge (z, y) .
- For each edge $(y, z) \in E(\vec{H})$ and integer $b \in \ell(\{y, z\}) \setminus \ell_1(\{y, z\})$, we choose a vertex $x \in V(G) \setminus \{y, z\}$ such that $(x, y), (x, z) \in E(\vec{F}_1)$ and $b = b_1 + b_2$ for some $b_1 \in \ell_1(\{x, y\})$ and $b_2 \in \ell_1(\{x, z\})$, and add the edge (y, x) . Note that such a vertex x and integers b_1 and b_2 exist, since b was added to $\ell(\{y, z\})$ when (\vec{F}, ℓ) was obtained from (\vec{F}_1, ℓ_1) as a 1-step fraternal augmentation.

Each edge $(y, x) \in E(\vec{H}_1) \setminus E(\vec{H})$ arises from an edge $(y, z) \in E(\vec{H})$ leaving y and an element $b \in \ell(\{y, z\}) \setminus \ell_1(\{y, z\})$, and each such pair contributes at most one edge leaving y . Hence, the maximum outdegree of \vec{H}_1 is at most $(r+1)\Delta$.

Consider an inwards-directed walk $(v_0 v_1 \dots v_t, b_1, \dots, b_t)$ of span b in \vec{H} , for any $b \leq r$. Then \vec{H} contains an inwards-directed walk of span b from v_0 to v_t obtained by natural edge replacements: For any edge $(y, z) \in E(\vec{H})$ of this walk and $b' \in \ell_i(\{y, z\})$, the construction described above ensures that if $(y, z) \notin E(\vec{H}_1)$ or $b' \notin \ell_1(\{y, z\})$, then there exists $x \in V(G) \setminus \{y, z\}$ such that $(y, x), (x, z) \in E(\vec{H}_1)$ and $b' = b'' + b'''$ for some $b'' \in \ell_1(\{x, y\})$ and $b''' \in \ell_1(\{x, z\})$, and we can replace the edge (y, z) in the walk by the edges (y, x) and (x, z) of $E(\vec{H}_1)$. Since \vec{H} represents $(\leq r)$ -distances in G , this transformation shows that so does \vec{H}_1 . ◀

We are now ready to prove the main result of this section.

► **Lemma 9.** *For any class \mathcal{G} with bounded expansion, there exists a function $d' : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ such that for each $G \in \mathcal{G}$ and each positive integer r , the graph G has an orientation with maximum outdegree at most $d'(r)$ that represents $(\leq r)$ -distances in G . Moreover, such an orientation can be found in time $O(r^2 d'(r) |V(G)|)$.*

Proof. Let d be the function from Lemma 5, and let $d'(r) = (r+1)^{r-1} d(r-1)$. By Lemma 6, we obtain an $(r-1)$ -step fraternal augmentation (\vec{H}, ℓ) of G of maximum outdegree at most $d(r-1)$. By Lemma 7, (\vec{H}, ℓ) represents $(\leq r)$ -distances in G . Repeatedly applying Lemma 8,

we obtain a 0-step fraternal supraugmentation (\vec{G}, ℓ_0) of G of maximum outdegree at most $d'(r)$ representing $(\leq r)$ -distances. Clearly, \vec{G} is an orientation of G of maximum outdegree at most $d'(r)$ representing $(\leq r)$ -distances. ◀

3 Approximation schemes

Let us now prove Theorem 2. To this end, let us start with a lemma to be applied to the sets arising from fractional treewidth-fragility.

► **Lemma 10.** *Let \vec{G} be an orientation of a graph G with maximum outdegree Δ . Let A be a set of vertices of G and for a positive integer c , let $\{R_v : v \in A\}$ be a system of subsets of A such that each vertex belongs to at most c of the subsets. For $X \subseteq V(G)$ and a positive integer r , let $D_{\vec{G},r}(X) \subseteq A$ be the union of the sets R_v for all vertices $v \in A$ such that \vec{G} contains a walk from v to X of length at most r directed away from v . For a positive integer k , let X_1, \dots, X_m be a system of subsets of $V(G)$ such that each vertex belongs to at most $\frac{m}{c(\Delta+1)^rk}$ of the subsets. For any assignment w of non-negative weights to vertices of G , there exists $i \in \{1, \dots, m\}$ such that $w(A \setminus D_{\vec{G},r}(X_i)) \geq (1 - 1/k)w(A)$.*

Proof. For a vertex $z \in A$, let $B(z)$ be the set of vertices reachable in \vec{G} from vertices $v \in A$ such that $z \in R_v$ by walks of length at most r directed away from v . Note that $|B(z)| \leq c(\Delta + 1)^r$ and that for each $X \subseteq V(G)$, we have $z \in D_{\vec{G},r}(X)$ if and only if $B(z) \cap X \neq \emptyset$.

Suppose for a contradiction that for each i we have $w(A \setminus D_{\vec{G},r}(X_i)) < (1 - 1/k)w(A)$, and thus $w(D_{\vec{G},r}(X_i)) > w(A)/k$. Then

$$\begin{aligned} \frac{m}{k}w(A) &< \sum_{i=1}^m w(D_{\vec{G},r}(X_i)) = \sum_{i=1}^m \sum_{z \in D_{\vec{G},r}(X_i)} w(z) = \sum_{i=1}^m \sum_{z \in A: B(z) \cap X_i \neq \emptyset} w(z) \\ &\leq \sum_{i=1}^m \sum_{z \in A} w(z) |B(z) \cap X_i| = \sum_{z \in A} w(z) \sum_{i=1}^m |B(z) \cap X_i| \\ &= \sum_{z \in A} w(z) \sum_{x \in B(z)} |\{i \in \{1, \dots, m\} : x \in X_i\}| \leq \sum_{z \in A} w(z) \sum_{x \in B(z)} \frac{m}{c(\Delta + 1)^rk} \\ &= \sum_{z \in A} w(z) |B(z)| \frac{m}{c(\Delta + 1)^rk} \leq \sum_{z \in A} w(z) \frac{m}{k} = \frac{m}{k}w(A), \end{aligned}$$

which is a contradiction. ◀

Next, let us derive a lemma on admissibility for $(\leq r)$ -distance determined problems.

► **Lemma 11.** *For a positive integer r , let \vec{G} be an orientation of a graph G representing $(\leq r)$ -distances. For a set $X \subseteq V(G)$, let $Y_{\vec{G},r}(X)$ be the set of vertices y such that \vec{G} contains a walk from y to X of length at most r directed away from y . For any $(\leq r)$ -distance determined problem, a set $B \subseteq V(G) \setminus Y_{\vec{G},r}(X)$ is admissible in G if and only if it is admissible in $G - X$.*

Proof. Since the problem is $(\leq r)$ -distance determined, it suffices to show that $\min(r, d_G(u, v)) = \min(r, d_{G-X}(u, v))$ holds for all $u, v \in B$. Clearly, $d_G(u, v) \leq d_{G-X}(u, v)$, and thus it suffices to show that if the distance between u and v in G is $b \leq r$, then $G - X$ contains a walk of length b between u and v . Since \vec{G} represents $(\leq r)$ -distances, there exists an inward-directed walk P of length b between u and v in \vec{G} . Since $u, v \notin Y_{\vec{G},r}(X)$, we have $V(P) \cap X = \emptyset$, and thus P is also a walk of length b between u and v in $G - X$. ◀

We are now ready to prove the main result.

Proof of Theorem 2. Let d' be the function from Lemma 9 for the class \mathcal{G} . Let us define $h(r, c) = c(d'(r) + 1)^r$. The algorithm is as follows. Since \mathcal{G} is q -efficiently fractionally treewidth- f -fragile, in time $q(|V(G)|)$ we can find sets $X_1, \dots, X_m \subseteq V(G)$ such that each vertex belongs to at most $\frac{m}{h(r, c)k}$ of them, and for each i , a tree decomposition of $G - X_i$ of width at most $f(h(r, c)k, |V(G)|)$. Clearly, $m \leq q(|V(G)|)$. Next, using Lemma 9, we find an orientation \vec{G} of G that represents $(\leq r)$ -distances. Let $Y_{\vec{G}, r}$ be defined as in the statement of Lemma 11. Since the problem is (g, p) -tw-tractable problem, for each i we can in time $p(|V(G)|) \cdot g(f(h(r, c)k, |V(G)|))$ find a subset A_i of $V(G) \setminus Y_{\vec{G}, r}(X_i)$ admissible in $G - X_i$ of largest weight. By Lemma 11, each of these sets is admissible in G ; the algorithm return the heaviest of the sets A_1, \dots, A_m .

As the returned set is admissible in G , it suffices to argue about its weight. Let A be a heaviest admissible set in G . Let $\{R_v \subseteq A : v \in A\}$ be the system of subsets from the definition of c -near-monotonicity, and let $D_{\vec{G}, r}$ be defined as in the statement of Lemma 10. By the definition of c -near-monotonicity, for each i the set $A \setminus D_{\vec{G}, r}(X_i)$ is admissible in G . Since $v \in R_v$ for each $v \in A$, we have $Y_{\vec{G}, r}(X_i) \cap A \subseteq D_{\vec{G}, r}(X_i)$, and thus $A \setminus D_{\vec{G}, r}(X_i) \subseteq V(G) \setminus Y_{\vec{G}, r}(X_i)$. By Lemma 11, $A \setminus D_{\vec{G}, r}(X_i)$ is also admissible in $G - X_i$, and by the choice of A_i , we have $w(A_i) \geq w(A \setminus D_{\vec{G}, r}(X_i))$. By Lemma 10, we conclude that for at least one i , we have $w(A_i) \geq (1 - 1/k)w(A)$, as required. ◀

References

- 1 Vladimir E. Alekseev, Vadim V. Lozin, Dmitriy S. Malyshev, and Martin Milanic. The maximum independent set problem in planar graphs. In *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2008. doi:10.1007/978-3-540-85238-4_7.
- 2 Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 293–299. ACM, 1990. doi:10.1145/100216.100254.
- 3 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.
- 4 Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 1999. doi:10.1007/3-540-48523-6_17.
- 5 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 6 Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Approximation schemes for first-order definable optimisation problems. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 411–420. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.13.
- 7 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and ptass. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 590–601. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070514>.

- 8 Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel P. Sanders, Bruce A. Reed, Paul D. Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *J. Comb. Theory, Ser. B*, 91(1):25–41, 2004. doi:10.1016/j.jctb.2003.09.001.
- 9 Zdenek Dvorák. Sublinear separators, fragility and subexponential expansion. *Eur. J. Comb.*, 52:103–119, 2016. doi:10.1016/j.ejc.2015.09.001.
- 10 Zdenek Dvorák. On classes of graphs with strongly sublinear separators. *Eur. J. Comb.*, 71:1–11, 2018. doi:10.1016/j.ejc.2018.02.032.
- 11 Zdenek Dvorák. Thin graph classes and polynomial-time approximation schemes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1685–1701. SIAM, 2018. doi:10.1137/1.9781611975031.110.
- 12 Zdenek Dvorák. Baker game and polynomial-time approximation schemes. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2227–2240. SIAM, 2020. doi:10.1137/1.9781611975994.137.
- 13 Zdenek Dvorák. Approximation metatheorem for fractionally treewidth-fragile graphs. *CoRR*, abs/2103.08698, 2021. arXiv:2103.08698.
- 14 Zdenek Dvorák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM J. Discret. Math.*, 30(2):1095–1101, 2016. doi:10.1137/15M1017569.
- 15 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.
- 16 Sarel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM J. Comput.*, 46(6):1712–1744, 2017. doi:10.1137/16M1079336.
- 17 Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Trans. Algorithms*, 2(3):335–363, 2006. doi:10.1145/1159892.1159895.
- 18 Robert Krauthgamer and James R. Lee. The intrinsic dimensionality of graphs. *Comb.*, 27(5):551–585, 2007. doi:10.1007/s00493-007-2183-y.
- 19 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979. doi:10.1137/0136016.
- 20 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046.
- 21 Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997. doi:10.1145/256292.256294.
- 22 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008. doi:10.1016/j.ejc.2006.07.014.
- 23 Jaroslav Nešetřil 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.

Modular and Submodular Optimization with Multiple Knapsack Constraints via Fractional Grouping

Yaron Fairstein ✉

Computer Science Department, Technion, Haifa, Israel

Ariel Kulik ✉

Computer Science Department, Technion, Haifa, Israel

Hadas Shachnai ✉

Computer Science Department, Technion, Haifa, Israel

Abstract

A multiple knapsack constraint over a set of items is defined by a set of bins of arbitrary capacities, and a weight for each of the items. An assignment for the constraint is an allocation of subsets of items to the bins which adheres to bin capacities. In this paper we present a unified algorithm that yields efficient approximations for a wide class of submodular and modular optimization problems involving multiple knapsack constraints. One notable example is a *polynomial time approximation scheme* for Multiple-Choice Multiple Knapsack, improving upon the best known ratio of 2. Another example is Non-monotone Submodular Multiple Knapsack, for which we obtain a $(0.385 - \epsilon)$ -approximation, matching the best known ratio for a single knapsack constraint. The robustness of our algorithm is achieved by applying a novel *fractional* variant of the classical linear grouping technique, which is of independent interest.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Sumodular Optimization, Multiple Knapsack, Randomized Rounding, Linear Grouping, Multiple Choice Multiple Knapsack

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.41

Related Version *Full Version*: <https://arxiv.org/abs/2007.10470>

1 Introduction

The Knapsack problem is one of the most studied problems in mathematical programming and combinatorial optimization, with applications ranging from power management and production planning, to blockchain storage allocation and key generation in cryptosystems [31, 26, 38, 41]. In a more general form, knapsack problems require assigning items of various sizes (weights) to a set of bins (knapsacks) of bounded capacities. The bin capacities then constitute the hard constraint for the problem. Formally, a *multiple knapsack constraint* (MKC) over a set of items is defined by a collection of bins of varying capacities and a non-negative weight for each item. A feasible solution for the constraint is an assignment of subsets of items to the bins, such that the total weight of items assigned to each bin does not exceed its capacity. This constraint plays a central role in the classic Multiple Knapsack problem [8, 23, 24]. The input is an MKC and each item also has a profit. The objective is to find a feasible solution for the MKC such that the total profit of assigned items is maximized.

Multiple Knapsack can be viewed as a maximization variant of the Bin Packing problem [25, 13]. In Bin Packing we are given a set of items, each associated with non-negative weight. We need to pack the items into a minimum number of identical (unit-size) bins.



© Yaron Fairstein, Ariel Kulik, and Hadas Shachnai;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, 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

A prominent technique for approximating Bin Packing is *grouping*, which decreases the number of distinct weights in the input instance. Informally, a subset of items is partitioned into groups G_1, \dots, G_τ , and all the items within a group are treated as if they have the *same* weight (e.g., [13, 25]). By properly forming the groups, the increase in the number of bins required for packing the instance can be bounded. Classic grouping techniques require knowledge of the items to be packed, and thus cannot be easily applied in the context of maximization problems, and specifically for a multiple knapsack constraint.

The main technical contribution of this paper is the introduction of *fractional grouping*, a variant of linear grouping which can be applied to multiple knapsack constraints. Fractional Grouping partitions the items into groups using an easy to obtain fractional solution, bypassing the requirement to know the items in the solution.

Fractional Grouping proved to be a robust technique for maximization problems. We use the technique to obtain, among others, a *polynomial-time approximation scheme (PTAS)* for the Multiple-Choice Multiple Knapsack Problem, a $(0.385 - \varepsilon)$ -approximation for non-monotone submodular maximization with a multiple knapsack constraint, and a $(1 - e^{-1} - o(1))$ -approximation for the Monotone Submodular Multiple Knapsack Problem with Uniform Capacities.

1.1 Problem Definition

We first define formally key components of the problem studied in this paper.

A *multiple knapsack constraint* (MKC) over a set I of items, denoted by $\mathcal{K} = (w, B, W)$, is defined by a weight function $w : I \rightarrow \mathbb{R}_{\geq 0}$, a set of bins B and bin capacities given by $W : B \rightarrow \mathbb{R}_{\geq 0}$. An *assignment* for the constraint is a function $A : B \rightarrow 2^I$ which assigns a subset of items to each bin. An assignment A is *feasible* if $\sum_{i \in A(b)} w(i) \leq W(b)$ for all $b \in B$. We say that A is an *assignment of S* if $S = \bigcup_{b \in B} A(b)$.

A set function $f : 2^I \rightarrow \mathbb{R}$ is *submodular* if for any $S \subseteq T \subseteq I$ and $i \in I \setminus T$ it holds that $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$.¹ Submodular functions naturally arise in numerous settings. While many submodular functions, such as coverage [19] and matroid rank function [6], are monotone, i.e., for any $S \subseteq T \subseteq I$, $f(S) \leq f(T)$, this is not always the case (cut functions [18] are a classic example). A special case of submodular functions is *modular* (or, *linear*) functions in which, for any $S \subseteq T \subseteq I$ and $i \in I \setminus T$, we have $f(S \cup \{i\}) - f(S) = f(T \cup \{i\}) - f(T)$.

For a constant $d \in \mathbb{N}$, the problem of *Submodular Maximization with d -Multiple Knapsack Constraints* (d -MKCP) is defined as follows. The input is $\mathcal{T} = (I, (\mathcal{K}_t)_{t=1}^d, \mathcal{I}, f)$, where I is a set of items, \mathcal{K}_t , $1 \leq t \leq d$ are d MKCs over I , $\mathcal{I} \subseteq 2^I$ and $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$ is a non-negative submodular function. \mathcal{I} is an additional constraint which can be one of the following: (i) $\mathcal{I} = 2^I$, i.e., any subset of items can be selected. (ii) \mathcal{I} is the independent set of a matroid,² or (iii) \mathcal{I} is the intersection of independent sets of two matroids, or (iv) \mathcal{I} is a matching.³ A solution for the instance is $S \in \mathcal{I}$ and $(A_t)_{t=1}^d$, where A_t is a feasible assignment of S w.r.t \mathcal{K}_t for $1 \leq t \leq d$. The value of the solution is $f(S)$, and the objective is to find a solution of maximal value.

We assume the function f is given via a value oracle. We further assume that the input indicates the type of constraint that \mathcal{I} represents. Finally, \mathcal{I} is given via a membership oracle, and if \mathcal{I} is a matroid intersection, a membership oracle is given for each matroid.

¹ Alternatively, for every $S, T \subseteq I$: $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$.

² A formal definition for matroid can be found in [34].

³ \mathcal{I} is a matching if there is a graph $G = (V, I)$, and $S \in \mathcal{I}$ iff S is a matching in G .

■ **Table 1** Results of Theorem 1 for d -MKCP.

Type of Additional Constraint	Modular Maximization	Monotone Submodular Max.	Non-Monotone Sub. Max
No additional constraint	PTAS	$1 - e^{-1} - \varepsilon$	$0.385 - \varepsilon$
Matroid constraint	PTAS	$1 - e^{-1} - \varepsilon$	—
2 matroids or a matching	PTAS	—	—

We refer to the special case in which f is monotone (modular) as monotone (modular) d -MKCP. Also, we use non-monotone d -MKCP when referring to general d -MKCP instances. Similarly, we refer to the special case in which \mathcal{I} is an independent set of a matroid (intersection of independent sets of two matroids or a matching) as d -MKCP with a matroid (matroid intersection or matching) constraint. If $\mathcal{I} = 2^I$ we refer to the problem as d -MKCP with no additional constraint. Thus, for example, in instances of modular 1-MKCP with a matroid constraint the function f is modular and \mathcal{I} is an independent set of a matroid.

Instances of d -MKCP naturally arise in various settings (see a detailed example in the full version of this paper [16]).

1.2 Our Results

Our main results are summarized in the next theorem (see also Table 1).

► **Theorem 1.** *For any fixed $d \in \mathbb{N}_+$ and $\varepsilon > 0$, there is*

1. *A randomized PTAS for modular d -MKCP $((1 - \varepsilon)$ -approximation). The same holds for this problem with a matroid constraint, matroid intersection constraint, or a matching constraint.*
2. *A polynomial-time random $(1 - e^{-1} - \varepsilon)$ -approximation for monotone d -MKCP with a matroid constraint.*
3. *A polynomial-time random $(0.385 - \varepsilon)$ -approximation for non-monotone d -MKCP with no additional constraint.*

All of the results are obtained using a single algorithm (Algorithm 2). The general algorithmic result encapsulates several important special cases. The Multiple-Choice Multiple Knapsack Problem is a variant of the Multiple Knapsack Problem in which the items are partitioned into classes C_1, \dots, C_k , and at most one item can be selected from each class. Formally, Multiple-Choice Multiple Knapsack is the special case of modular 1-MKCP where \mathcal{I} describes a partition matroid.⁴ The problem has natural applications in network optimization [12, 37]. The best known approximation ratio for the problem is 2 due to [12]. This approximation ratio is improved by Theorem 1, as stated in the following.

► **Corollary 2.** *There is a randomized PTAS for the Multiple-Choice Multiple Knapsack Problem.*

While the Multiple Knapsack Problem and the Monotone Submodular Multiple Knapsack Problem are well understood [8, 23, 24, 15, 35], no results were previously known for the Non-Monotone Submodular Multiple Knapsack Problem, the special case of non-monotone 1-MKCP with no additional constraint. A constant approximation ratio for the problem is obtained as a special case of Theorem 1.

⁴ That is, $\mathcal{I} = \{S \subseteq I \mid \forall 1 \leq j \leq k : |S \cap C_j| \leq 1\}$ where C_1, \dots, C_k is a partition of I .

► **Corollary 3.** *For any $\varepsilon > 0$ there is a polynomial time random $(0.385 - \varepsilon)$ -approximation for the Non-Monotone Submodular Multiple Knapsack Problem.*

A PTAS for Multistage Multiple Knapsack, a multistage version of the Multiple Knapsack Problem, can be obtained via a reduction to modular d -MKCP with a matroid constraint.⁵ Here, to obtain a $(1 - O(\varepsilon))$ -approximation for the multistage problem, the reduction solves instances of modular $\Theta(\frac{1}{\varepsilon})$ -MKCP with a matroid constraint (see [14] for details). Beyond the rich set of applications, our ability to derive such a general result is an evidence for the robustness of fractional grouping, the main technical contribution of this paper.

Our result for modular d -MKCP, for $d \geq 2$, generalizes the PTAS for the classic d -dimensional Knapsack problem ($\mathcal{I} = 2^I$ and $|B_t| = 1$ for any $1 \leq t \leq d$). Furthermore, a PTAS is the best we can expect as there is no *efficient PTAS (EPTAS)* already for d -dimensional Knapsack, unless $W[1] = \text{FPT}$ [28]. While there is a well-known PTAS for Multiple Knapsack [8], existing techniques do not readily enable handling additional constraints, such as a matroid constraint.

The approximation ratio obtained for monotone d -MKCP is nearly optimal, as for any $\varepsilon > 0$ there is no $(1 - e^{-1} + \varepsilon)$ -approximation for monotone submodular maximization with a cardinality constraint in the oracle model [32]. The approximation ratio is also tight under $P \neq NP$ due to the special case of coverage functions [19]. Previous works [15, 35] obtained the same approximation ratio for the Monotone Submodular Multiple Knapsack Problem (i.e., monotone 1-MKCP). However, as in the modular case, existing techniques are limited to handling a single MKC (with no other constraints).

In the non-monotone case, the approximation ratio is in fact $(c - \varepsilon)$ for any $\varepsilon > 0$, where $c > 0.385$ is the ratio derived in [4]. This approximation ratio matches the current best known ratio for non-monotone submodular maximization with a single knapsack constraint [4]. A 0.491 hardness of approximation bound for non-monotone d -MKCP follows from [22].

Our technique can be cast also as a variant of *contention resolution scheme* [11]. The scheme can be used to derive approximation algorithms for special cases of d -MKCP which are not considered in Theorem 1. Such a scheme can be found in an earlier version of this paper [17].⁶

The *Monotone Submodular Multiple Knapsack Problem with Uniform Capacities* (USMKP) is the special case of d -MKCP in which $\mathcal{I} = 2^I$, $d = 1$, f is monotone, and furthermore, all the bins in the MKC have the same capacity. That is, $\mathcal{K}_1 = (w, B, W)$ and $W(b_1) = W(b_2)$ for any $b_1, b_2 \in B$. This restricted variant of d -MKCP commonly arises in real-life applications (e.g., in file assignment to several identical storage devices). The best known approximation ratio for USMKP is $(1 - e^{-1} - \varepsilon)$ for any fixed $\varepsilon > 0$ [15, 35]. Another contribution of this paper is an improvement of this ratio.

► **Theorem 4.** *There is a polynomial-time random $(1 - e^{-1} - O((\log |B|)^{-\frac{1}{4}}))$ -approximation for the Monotone Submodular Multiple Knapsack Problem with Uniform Capacities.*

1.3 Related Work

In the classic Multiple Knapsack problem, the goal is to maximize a modular set function subject to a single multiple knapsack constraint. A PTAS for the problem was first presented by Chekuri and Khanna [8]. The authors also ruled out the existence of a *fully polynomial time approximation scheme (FPTAS)*. An EPTAS was later developed by Jansen [23, 24].

⁵ See, e.g., [2] for the Multistage Knapsack model.

⁶ We were unable to obtain tight approximation ratios for the studied problems using this approach.

In the Bin Packing problem, we are given a set I of items, a weight function $w : I \rightarrow \mathbb{R}_{\geq 0}$ and a capacity $W > 0$. The objective is to partition the set I into a minimal number of sets S_1, \dots, S_m (i.e., find a *packing*) such that $\sum_{i \in S_b} w(i) \leq W$ for all $1 \leq b \leq m$. In [25] the authors presented a polynomial-time algorithm which returns a packing using $\text{OPT} + O(\log^2 \text{OPT})$ bins, where OPT is the number of bins in a minimal packing. The result was later improved by Rothvoß [33].

Research work on monotone submodular maximization dates back to the late 1970's. In [32] Nemhauser and Wolsey presented a greedy-based tight $(1 - e^{-1})$ -approximation for maximizing a monotone submodular function subject to a cardinality constraint, along with a matching lower bound in the oracle model. The greedy algorithm of [32] was later generalized to monotone submodular maximization subject to a knapsack constraint [27, 36].

A major breakthrough in the field of submodular optimization resulted from the introduction of algorithms for optimizing the *multilinear extension* of a submodular function ([6, 30, 7, 40, 20, 5]). For $\bar{x} \in [0, 1]^I$, we say that a random set $S \subseteq I$ is distributed by \bar{x} (i.e., $S \sim \bar{x}$) if $\Pr(i \in S) = \bar{x}_i$, and the events $(i \in S)_{i \in I}$ are independent. Given a function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$, its *multilinear extension* is $F : [0, 1]^I \rightarrow \mathbb{R}_{\geq 0}$ defined as $F(\bar{x}) = \mathbb{E}_{S \sim \bar{x}}[f(S)]$.

The input for the problem of optimizing the multilinear relaxation is an oracle for a submodular function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$ and a downward closed solvable polytope P .⁷ The objective is to find $\bar{x} \in P$ such that $F(\bar{x})$ is maximized, where F is the multilinear extension of f . The problem admits a random $(1 - e^{-1} - o(1))$ -approximation in the monotone case and a random $(0.385 + \delta)$ -approximation in the non-monotone case (for some small constant $\delta > 0$) due to [7] and [4].

Several techniques were developed for *rounding* a (fractional) solution for the multilinear optimization problem to an integral solution. These include Pipage Rounding [1], Randomized Swap Rounding [9], and Contention Resolution Schemes [11]. These techniques led to the state of art results for many problems (e.g., [29, 7, 1, 9]).

A random $(1 - e^{-1} - \varepsilon)$ -approximation for the Monotone Submodular Multiple Knapsack problem was presented in [15]. The technique in [15] modifies the objective function and its domain. This modification does not preserve submodularity of a non-monotone function and the combinatorial properties of additional constraints. Thus, it does not generalize to d -MKCP.

A deterministic $(1 - e^{-1} - \varepsilon)$ -approximation for Monotone Submodular Multiple Knapsack was later obtained by Sun et al. [35]. Their algorithm relies on a variant of the greedy algorithm of [36] which cannot be extended to the non-monotone case, or easily adapted to handle more than a single MKC.

1.4 Technical Overview

In the following we describe the technical problem solved by fractional grouping and give some insight to the way we solve this problem. For simplicity, we focus on the special case of 1-MKCP, in which the number of bins is large and all bins have unit capacity. Let $(I, (w, B, W), 2^I, f)$ be a 1-MCKP instance where $W(b) = 1$ for all $b \in B$. Also, assume that no two items have the same weight. Let S^* and A^* be an optimal solution for the instance.

⁷ A polytope $P \in [0, 1]^I$ is *downward closed* if for any $\bar{x} \in P$ and $\bar{y} \in [0, 1]^I$ such that $\bar{y} \leq \bar{x}$ (that is, $\bar{y}_i \leq \bar{x}_i$ for every $i \in I$) it holds that $\bar{y} \in P$. A polytope $P \in [0, 1]^I$ is *solvable* if, for any $\bar{\lambda} \in \mathbb{R}^I$, a point $\bar{x} \in P$ such that $\bar{\lambda} \cdot \bar{x} = \max_{\bar{y} \in P} \bar{\lambda} \cdot \bar{y}$ can be computed in polynomial time, where $\bar{\lambda} \cdot \bar{x}$ is the dot product of $\bar{\lambda}$ and \bar{x} .

Fix an arbitrary small $\mu > 0$ such that $\mu^{-2} \in \mathbb{N}$. We say that an item $i \in I$ is *heavy* if $w(i) > \mu$; otherwise, i is *light*. Let $H \subseteq I$ denote the heavy items. We can apply linear grouping [13] to the heavy items in S^* . That is, let $h^* = |S^* \cap H|$ be the number of heavy items in S^* , and partition $S^* \cap H$ to μ^{-2} groups of cardinality $\mu^2 \cdot h^*$, assuming the items are sorted in decreasing order by weights (for simplicity, assume $h^* \geq \mu^{-2}$ and $\mu^2 \cdot h^* \in \mathbb{N}$). Specifically, $S^* \cap H = G_1^* \cup \dots \cup G_{\mu^{-2}}^*$, where $|G_k^*| = \mu^2 \cdot h^*$ for all $1 \leq k \leq \mu^{-2}$ and for any $i_1 \in G_{k_1}^*, i_2 \in G_{k_2}^*$ where $k_1 < k_2$ we have that $w(i_1) > w(i_2)$. Also, for any $1 \leq k \leq \mu^{-2}$ let q_k , the k -th pivot, be the item of highest weight in G_k^* .

We use the pivots to generate a new collection of groups $G_1, \dots, G_{\mu^{-2}}$ where $G_k = \{i \in H \mid w(q_{k+1}) < w(i) \leq w(q_k)\}$ for $1 \leq k < \mu^{-2}$, and $G_{\mu^{-2}} = \{i \in H \mid w(i) \leq w(q_{\mu^{-2}})\}$. Clearly, $G_k^* \subseteq G_k$ for any $1 \leq k \leq \mu^{-2}$. Let $X = \{i \in H \mid w(i) > w(q_1)\}$ be the set of largest items in H .

A standard *shifting* argument can be used to show that any set $S \subseteq I \setminus X$, such that $w(S) \leq |B|$ and $|S \cap G_k| \leq \mu^2 \cdot h^*$ for all $1 \leq k \leq \mu^{-2}$, can be packed into $(1 + 2\mu)|B| + 1$ bins as follows.⁸ The items in $S \cap G_k$ can be packed in place of the items in G_{k-1}^* in A^* , each of the items in $S \cap G_1$ can be packed in a separate bin (observe that $|S \cap G_1| \leq \mu^2 \cdot h^* \leq \mu|B|$ as packing of h^* heavy items requires at least $h^* \cdot \mu$ bins). Finally, First-Fit can be used to pack the light items in S .

Now, assume we know $q_1, \dots, q_{\mu^{-2}}$ and h^* ; thus, the sets $G_1, \dots, G_{\mu^{-2}}$ and X can be constructed. Consider the following optimization problem: find $S \subseteq I \setminus X$ such that $w(S) \leq |B|$, $|S \cap G_k| \leq \mu^2 \cdot h^*$ for all $1 \leq k \leq \mu^{-2}$, and $f(S)$ is maximal. The problem is an instance of non-monotone submodular maximization with a $(1 + \mu^{-2})$ -dimensional knapsack constraint, for which there is a $(0.385 - \epsilon)$ -approximation algorithm [29, 4]. The algorithm can be used to find $S \subseteq I \setminus X$ which satisfies the above constraints and $f(S) \geq (0.385 - \epsilon) \cdot f(S^*)$, as S^* is a feasible solution for the problem. Subsequently, S can be packed into bins using a standard bin packing algorithm. This will lead to a packing of S into roughly $(1 + 2\mu)|B| + O(\log^2 |B|)$ bins. By removing the bins of least value (along with their items), and using the assumption that $|B|$ is sufficiently large, we can obtain a set S' and an assignment of S' into B such that $f(S)$ is arbitrarily close to $0.385 \cdot f(S^*)$.

Indeed, we do not know the values of $q_1, \dots, q_{\mu^{-2}}$ and h^* . This prevents us from using the above approach. However, as in [3], we can overcome this difficulty through exhaustive enumeration. Each of $q_1, \dots, q_{\mu^{-2}}$ and h^* takes one of $|I|$ possible values. Thus, by iterating over all $|I|^{1+\mu^{-2}}$ possible values for $q_1, \dots, q_{\mu^{-2}}$ and h^* , and solving the above problem for each, we can find a solution of value at least $0.385 \cdot f(S^*)$.

While this approach is useful for our restricted class of instances, due to the use of exhaustive enumeration it does not scale to general instances, where bin capacities may be arbitrary. Known techniques ([15]) can be used to reduce the number of unique bin capacities in a general MKC to be logarithmic in $|B|$. As enumeration is required for each unique capacity, this results in $|I|^{\Theta(\log |B|)}$ iterations, which is non-polynomial.

Fractional Grouping overcomes this hurdle by using a polytope $P \subseteq [0, 1]^I$ to represent an MKC. A grouping $G_1^{\bar{y}}, \dots, G_\tau^{\bar{y}}$ with $\tau \leq \mu^{-2} + 1$ is derived from a vector $\bar{y} \in P$. The polytope P bears some similarity to configuration linear programs used in previous works ([24, 21, 3]). While P is not solvable, it satisfies an approximate version of solvability which suffices for our needs.

Fractional grouping satisfies the main properties of the grouping defined for S^* . Each of the groups contains roughly the same number of fractionally selected items. That is, $\sum_{i \in G_k^{\bar{y}}} \bar{y}_i \approx \mu^2 |B|$ for all $1 \leq k \leq \tau$. Furthermore, we show that if \bar{y} is strictly contained in

⁸ For a set $S \subseteq I$ we denote $w(S) = \sum_{i \in S} w(i)$.

P then any subset $S \subseteq I$ satisfying (i) $|S \cap G_k| \leq \mu|B|$ for all $1 \leq k \leq \tau$, and (ii) $w(S \setminus H)$ is sufficiently small, can be packed into strictly less than $|B|$ bins (see the details in Section 2). The existence of a packing for S relies on a shifting argument similar to the one used above. In this case, however, the structure of the polytope P replaces the role of S^* in our discussion.

This suggests the following algorithm. Use the algorithm of [4] to find $\bar{y} \in P$ such that $F(\bar{y}) \geq (0.385 - \varepsilon)f(S^*)$, and sample a random set $R \sim (1 - \delta)^2 \bar{y}$. By the above property, R can be packed into strictly less than $|B|$ bins with high probability, as $\mathbb{E}[|R \cap G_k|] \ll \mu|B|$. Thus, R can be packed into B using a bin packing algorithm. Standard tools (specifically, the FKG inequality as used in [11]) can also be used to show that $\mathbb{E}[f(R)]$ is arbitrarily close to $F(\bar{y})$. Hence, we can obtain an approximation ratio arbitrarily close to $(0.385 - \varepsilon)$ while avoiding enumeration.

This core idea of fractional grouping for bins of uniform capacities can be scaled to obtain Theorem 1. This scaling involves use of existing techniques for submodular optimization ([15, 9, 10, 7, 4]), along with a novel *block association* technique we apply to handle MKCs with arbitrary bin capacities.

Organization

We present the fractional grouping technique in Section 2. Our algorithms for uniform bin capacities and the general case are given in Section 3 and 4, respectively. Due to space constraints, the block association technique and some proofs are omitted. Those appear in the full version [16].

2 Fractional Grouping

Given an MKC (w, B, W) over I , a subset of bins $K \subseteq B$ is a *block* if all the bins in K have the same capacity. Denote by W_K^* the capacities of the bins in block K , then $W_K^* = W(b)$ for any $b \in K$.

We first define a polytope P_K which represents the block $K \subseteq B$ of an MKC (w, B, W) over I . To simplify the presentation, we assume the MKC (w, B, W) and K are fixed throughout this section. W.l.o.g., assume that $I = \{1, 2, \dots, n\}$ and $w(1) \geq w(2) \geq \dots \geq w(n)$. A *K-configuration* is a subset $C \subseteq I$ of items which fits into a single bin of block K , i.e., $w(C) \leq W_K^*$. We use \mathcal{C}_K to denote the set of all K -configurations. Formally, $\mathcal{C}_K = \{C \subseteq I \mid w(C) \leq W_K^*\}$.

► **Definition 5.** *The extended block polytope of K is*

$$P_K^e = \left\{ \bar{y} \in [0, 1]^I, \bar{z} \in [0, 1]^{\mathcal{C}_K} \mid \forall i \in I : \begin{array}{l} \sum_{C \in \mathcal{C}_K} \bar{z}_C \leq |K| \\ \bar{y}_i \leq \sum_{\substack{C \in \mathcal{C}_K \\ i \in C}} \bar{z}_C \end{array} \right\} \quad (1)$$

The first constraint in (1) bounds the number of selected configurations by the number of bins. The second constraint requires that each selected item is (fractionally) covered by a corresponding set of configurations. It is easy to verify that, for any $(\bar{y}, \bar{z}) \in P_K^e$, it holds that $\sum_{i \in I} w(i) \cdot \bar{y}_i \leq |K| \cdot W_K^*$.

► **Definition 6.** *The block polytope of K is*

$$P_K = \{ \bar{y} \in [0, 1]^I \mid \exists \bar{z} \in [0, 1]^{\mathcal{C}_K} : (\bar{y}, \bar{z}) \in P_K^e \}. \quad (2)$$

While P_K^e and P_K are defined using an exponential number of variables (as $\bar{z} \in [0, 1]^{C_K}$ and C_K is exponential), it follows from standard arguments (see, e.g., [21, 25]) that, for any $\bar{c} \in \mathbb{R}^I$, $\max_{\bar{y} \in P_K} \bar{c} \cdot \bar{y}$ can be approximated.

► **Lemma 7.** *There is a fully polynomial-time approximation scheme (FPTAS) for the problem of finding $\bar{y} \in P_K$ such that $\bar{c} \cdot \bar{y}$ is maximized, given an MKC (w, B, W) , a block $K \subseteq B$ and a vector $\bar{c} \in \mathbb{R}^I$, where P_K is the block polytope of K .*

A formal proof for Lemma 7 is given in [16]. We say that $A : K \rightarrow 2^I$ is a *feasible assignment* for K if $w(A(b)) \leq W_K^*$ for any $b \in K$. Also, we use $\mathbb{1}_S = \bar{x} \in \{0, 1\}^I$, where $\bar{x}_i = 1$ if $i \in S$ and $\bar{x}_i = 0$ if $i \in I \setminus S$. The next lemma implies that the definition of P_K^e is sound for the problem.

► **Lemma 8.** *Let A be a feasible assignment for K and $S = \bigcup_{b \in K} A(b)$. Then $\mathbb{1}_S \in P_K$.*

The lemma is easily proved, by setting $\bar{z}_C = 1$ if $A(b) = C$ for some $b \in B$, and $\bar{z}_C = 0$ otherwise. We say an item $i \in I$ is μ -heavy for $\mu > 0$ (w.r.t K) if $W_K^* \geq w(i) > \mu \cdot W_K^*$; $i \in I$ is μ -light if $w(i) \leq \mu W_K^*$. Denote by $H_{K,\mu}$ and $L_{K,\mu}$ the sets of μ -heavy items and μ -light items, respectively.

Given a vector $\bar{y} \in P_K$, we now describe the partition of μ -heavy items into groups G_1, \dots, G_τ , for some $\tau \leq \mu^{-2} + 1$. Starting with $k = 1$ and $G_k = \emptyset$, add items from $H_{K,\mu}$ to the current group G_k until $\sum_{i \in G_k} \bar{y}_i \geq \mu |K|$. Once the constraint is met, mark the index of the last item in G_k as q_k , the μ -pivot of G_k , close G_k and open a new group, G_{k+1} . Each of the groups $G_1, \dots, G_{\tau-1}$ represents a fractional selection of $\approx \mu |K|$ heavy items of \bar{y} . The last group, G_τ , contains the remaining items in $H_{K,\mu}$, for which the μ -pivot is q_{\max} (last item in $H_{K,\mu}$). We now define formally the partition process.

► **Definition 9.** *Let $\bar{y} \in P_K$ and $\mu \in (0, \frac{1}{2}]$. Also, let $q_0 \in \{0, 1, \dots, n\}$ and $q_{\max} \in I$ such that $H_{K,\mu} = \{i \in I \mid q_0 < i \leq q_{\max}\}$. The μ -pivots of \bar{y} , given by q_1, \dots, q_τ , are defined inductively, i.e.,*

$$q_k = \min \left\{ s \in H_{K,\mu} \mid \sum_{i=q_{k-1}+1}^s \bar{y}_i \geq \mu \cdot |K| \right\}.$$

If the set over which the minimum is taken is empty, let $\tau = k$ and $q_\tau = q_{\max}$. The μ -grouping of \bar{y} consists of the sets G_1, \dots, G_τ , where $G_k = \{i \in H_{K,\mu} \mid q_{k-1} < i \leq q_k\}$ for $1 \leq k \leq \tau$.

Note that in the above definition it may be that $q_0 \neq 0$ as there may be items $i \in I$ for which $w(i) > W_K^*$. Given a polytope P and $\delta \in \mathbb{R}$, we use the notation $\delta P = \{\delta \bar{x} \mid \bar{x} \in P\}$. The main properties of fractional grouping are summarized in the next lemma.

► **Lemma 10 (Fractional Grouping).** *For any $\bar{y} \in P_K$ and $0 < \mu < \frac{1}{2}$ there is a polynomial time algorithm which computes a partition G_1, \dots, G_τ of $H_{K,\mu}$ with $\tau \leq \mu^{-2} + 1$ for which the following hold:*

1. $\sum_{i \in G_k} \bar{y}_i \leq \mu \cdot |K| + 1$ for any $1 \leq k \leq \tau$.
2. Let $S \subseteq H_{K,\mu} \cup L_{K,\mu}$ such that $|S \cap G_k| \leq \mu |K|$ for every $1 \leq k \leq \tau$, and $w(S \cap L_{K,\mu}) \leq \sum_{i \in L_{K,\mu}} \bar{y}_i \cdot w(i) + \lambda \cdot W_K^*$ for some $\lambda \geq 0$. Also, assume $\bar{y} \in (1 - \delta)P_K$ for some $\delta \geq 0$. Then S can be packed into $(1 - \delta + 3\mu)|K| + 4 \cdot 4^{\mu^{-2}} + 2\lambda$ bins of capacity W_K^* .

We refer to G_1, \dots, G_τ as the μ -grouping of \bar{y} .

Proof. It follows from Definition 9 that G_1, \dots, G_τ can be computed in polynomial time. Also, $\sum_{i \in G_\tau} \bar{y}_i < \mu \cdot |K|$ and

$$\forall 1 \leq k < \tau : \quad \mu \cdot |K| \leq \sum_{i \in G_k} \bar{y}_i \leq \mu \cdot |K| + 1. \quad (3)$$

Furthermore, $\tau \leq \mu^{-2} + 1$. Thus, it remains to show Property 2 in the lemma.

Define the *type* of a configuration $C \in \mathcal{C}_K$, denoted by $\text{type}(C)$, as the vector $T \in \mathbb{N}^\tau$ with $T_k = |C \cap G_k|$. Let $\mathcal{T} = \{\text{type}(C) \mid C \in \mathcal{C}_K\}$ be the set of all types. Given a type $T \in \mathcal{T}$, consider a set of items $Q \subseteq H_{K,\mu} \setminus G_1$, such that $|Q \cap G_k| \leq T_{k-1}$ for any $2 \leq k \leq \tau$, then $w(Q) \leq W_K^*$. This is true since we assume the items in $H_{K,\mu}$ are sorted in non-increasing order by weights. We use this key property to construct a packing for S .

We note that $\sum_{k=1}^\tau |C \cap G_k| < \mu^{-1}$ for any $C \in \mathcal{C}_K$ (otherwise $w(C) > W_K^*$, as $G_k \subseteq H_{K,\mu}$). It follows that $|\mathcal{T}| \leq 4^{\mu^{-2}}$. Indeed, the number of types is bounded by the number of different non-negative integer τ -tuples whose sum is at most μ^{-1} .

By Definition 5, there exists $\bar{z} \in [0, 1]^{\mathcal{C}_K}$ such that $(\bar{y}, \bar{z}) \in (1 - \delta)P_K^e$. For $T \in \mathcal{T}$, let $\eta(T) = \sum_{C \in \mathcal{C}_K \text{ s.t. } \text{type}(C)=T} \bar{z}_C$. Then, for any $1 \leq k \leq \tau - 1$, we have

$$\mu |K| \leq \sum_{i \in G_k} \bar{y}_i \leq \sum_{i \in G_k} \sum_{C \in \mathcal{C}_K \text{ s.t. } i \in C} \bar{z}_C = \sum_{C \in \mathcal{C}_K} |G_k \cap C| \bar{z}_C = \sum_{T \in \mathcal{T}} T_k \cdot \eta(T) \quad (4)$$

The first inequality follows from (3). The second inequality follows from (1). The two equalities follow by rearranging the terms.

Using \bar{z} (through the values of $\eta(T)$) we define an assignment of $S \cap (G_2 \cup \dots \cup G_\tau)$ to $\eta = \sum_{T \in \mathcal{T}} \lceil \eta(T) \rceil$ bins. We initialize η sets (bins) $A_1, \dots, A_\eta = \emptyset$ and associate a type with each set A_b , such that there are $\lceil \eta(T) \rceil$ sets associated with the type $T \in \mathcal{T}$, using a function R . That is, let $R : \{1, 2, \dots, \eta\} \rightarrow \mathcal{T}$ such that $|R^{-1}(T)| = \lceil \eta(T) \rceil$. We assign the items in $S \cap (G_2 \cup \dots \cup G_\tau)$ to A_1, \dots, A_η while ensuring that $|A_b \cap G_k| \leq R(b)_{k-1}$ for any $1 \leq b \leq \eta$ and $2 \leq k \leq \tau$. In other words, the number of items assigned to A_b from G_k is at most the number of items from G_{k-1} in the configuration type T assigned to bin b by R . The assignment is obtained as follows. For every $2 \leq k \leq \tau$, iterate over the items $i \in S \cap G_k$, find $1 \leq b \leq \eta$ such that $|A_b \cap G_k| < R(b)_{k-1}$ and set $A_b \leftarrow A_b \cup \{i\}$. It follows from (4) and the conditions of the lemma that such b will always be found.

Upon completion of the process, we have that $S \cap (G_2 \cup \dots \cup G_\tau) = A_1 \cup \dots \cup A_\eta$. Furthermore, for every $1 \leq b \leq \eta$, there are $C \in \mathcal{C}_K$ and $T \in \mathcal{T}$ such that $\text{type}(C) = T = R(b)$. Since $A_b \subseteq G_2 \cup \dots \cup G_\tau$, we have

$$w(A_b) = \sum_{k=2}^\tau w(A_b \cap G_k) \leq \sum_{k=2}^\tau T_{k-1} \cdot w(q_{k-1}) = \sum_{k=2}^\tau |C \cap G_{k-1}| \cdot w(q_{k-1}) \leq \sum_{i \in C} w(i) \leq W_K^*.$$

The first inequality holds since $w(q_{k-1}) \geq w(i)$ for every $i \in G_k$, and the second holds since $w(q_{k-1}) \leq w(i)$ for every $i \in G_{k-1}$. By similar arguments, for every $2 \leq k \leq \tau$, we have

$$w(S \cap G_k) \leq |S \cap G_k| \cdot w(q_{k-1}) \leq \mu |K| \cdot w(q_{k-1}) \leq \sum_{i \in G_{k-1}} \bar{y}_i \cdot w(q_{k-1}) \leq \sum_{i \in G_{k-1}} \bar{y}_i \cdot w(i). \quad (5)$$

The third inequality is due to (3). Using (5) and the conditions in the lemma,

$$\begin{aligned} w(S \setminus G_1) &= w(S \cap L_{K,\mu}) + \sum_{k=2}^\tau w(S \cap G_k) \leq \sum_{i \in L_{K,\mu}} \bar{y}_i w(i) + \lambda W_K^* + \sum_{k=1}^{\tau-1} \sum_{i \in G_k} \bar{y}_i w(i) \\ &\leq \sum_{i \in I} \bar{y}_i \cdot w(i) + \lambda W_K^* \leq (1 - \delta) W_K^* \cdot |K| + \lambda W_K^*. \end{aligned} \quad (6)$$

We use First-Fit (see, e.g., Chapter 9 in [39]) to add the items in $S \cap L_{K,\mu}$ to the sets (=bins) A_1, \dots, A_η while maintaining the capacity constraint, $w(A_b) \leq W_K^*$. First-Fit iterates over the items $i \in S \cap L_{K,\mu}$ and searches for a minimal b such that $w(A_b \cup \{i\}) \leq W_K^*$. If such b exists, First-Fit updates $A_b \leftarrow A_b \cup \{i\}$; otherwise, it adds a new bin with i as its content. Let η' be the number of bins by the end of the process. As $w(i) \leq \mu W_K^*$ for $i \in S \cap L_{K,\mu}$, and due to (6), it holds that $\eta' \leq \max\{\eta, (|K|(1-\delta) + \lambda)(1+2\mu) + 1\}$. Finally,

$$\eta = \sum_{T \in \mathcal{T}} \lceil \eta(T) \rceil \leq |\mathcal{T}| + \sum_{T \in \mathcal{T}} \eta(T) \leq 4^{\mu-2} + \sum_{C \in \mathcal{C}_K} \bar{z}_C \leq 4^{\mu-2} + (1-\delta)|K|.$$

Thus, there is a packing of $S \setminus G_1$ into at most $(1-\delta)|K| + 4^{\mu-2} + 1 + 2\mu|K| + 2\lambda$ bins of capacity W_K^* . Since $|S \cap G_1| \leq \mu|K|$, each of the items in $S \cap G_1$ can be packed into a bin of its own. This yields a packing using at most $(1-\delta + 3\mu)|K| + 4 \cdot 4^{\mu-2} + 2\lambda$ bins. ◀

3 Uniform Capacities

In this section we apply fractional grouping (as stated in Lemma 10) to solve the Monotone Submodular Multiple Knapsack Problem with Uniform Capacities (USMKP). An instance of the problem consists of an MKC (w, B, W) over a set I of items, such that $W_B^* = W(b)$ for all $b \in B$, and a submodular function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$. For simplicity, we associate a solution for the problem with a feasible assignment $A : B \rightarrow 2^I$. Then, the set of assigned items is given by $S = \bigcup_{b \in B} A(b)$.

Our algorithm for USMKP instances applies *Pipage Rounding* [1, 6]. The input for a Pipage Rounding step is a (fractional) solution $\bar{x} \in [0, 1]^I$, and two items $i_1, i_2 \in I$ with costs $c_1, c_2 \geq 0$. The Pipage Rounding step returns a new random solution $\bar{x}' \in [0, 1]^I$ such that $\bar{x}'_i = \bar{x}_i$ for $i \in I \setminus \{i_1, i_2\}$, $\bar{x}'_{i_1} \cdot c_1 + \bar{x}'_{i_2} \cdot c_2 = \bar{x}_{i_1} \cdot c_1 + \bar{x}_{i_2} \cdot c_2$, and either $\bar{x}'_{i_1} \in \{0, 1\}$ or $\bar{x}'_{i_2} \in \{0, 1\}$. Furthermore, for any submodular function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$ it holds that $\mathbb{E}[F(\bar{x}')] \geq F(\bar{x})$, where F is the multilinear extension of f . Algorithm 1 calls a subroutine $\text{Pipage}(\bar{x}, f, G, \bar{c})$, which can be implemented by an iterative application of Pipage Rounding steps as long as \bar{x} contains two fractional entries, and randomly sampling the last remaining fractional entry. The properties of **Pipage** are summarized in the next result.

► **Lemma 11.** *There is a polynomial time procedure $\text{Pipage}(\bar{x}, f, G, \bar{c})$ for which the following holds. Given $\bar{x} \in [0, 1]^I$, a submodular function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$, a subset of items $G \subseteq I$ and a cost vector for the items $\bar{c} \in \mathbb{R}_{\geq 0}^G$, the procedure returns a random vector $\bar{x}' \in [0, 1]^I$ such that $\mathbb{E}[F(\bar{x}')] \geq F(\bar{x})$, $\bar{x}'_i \in \{0, 1\}$ for $i \in G$, $\bar{x}'_i = \bar{x}_i$ for all $i \in I \setminus G$, and there is $i^* \in G$ such that $\sum_{i \in G} \bar{x}'_i \cdot c_i \leq c_{i^*} + \sum_{i \in G} \bar{x}_i \cdot c_i$.*

To solve USMKP instances, our algorithm initially finds $\bar{y} \in P_B$, where P_B is the block polytope of B (note that B is a block in this case), for which $F(\bar{y})$ is large (F is the multilinear extension of the value function f). The algorithm chooses a small value for μ and uses G_1, \dots, G_τ , the μ -grouping of $(1-4\mu)\bar{y}$, to guide the rounding process. Pipage rounding is used to convert $(1-4\mu) \cdot \bar{y}$ to $S \subseteq I$ while preserving the number of selected items from each group as $\approx \mu|B|$, and the total weight of items selected from $L_{B,\mu}$ (i.e., μ -light items) as $\approx (1-4\mu) \cdot \sum_{i \in L_{B,\mu}} \bar{y}_i \cdot w(i)$. An approximation algorithm for bin packing is then used to find a packing of S to the bins. Lemma 10 ensures the resulting packing uses at most $|B|$ bins for sufficiently large B . In case the packing requires more than $|B|$ bins we simply assume the algorithm returns an empty solution. We give the pseudocode in Algorithm 1.

► **Lemma 12.** *Algorithm 1 yields a $\left(1 - e^{-1} - O\left((\log |B|)^{-\frac{1}{4}}\right)\right)$ -approximation for USMKP.*

■ **Algorithm 1** Submodular Multiple Knapsack with Uniform Capacities.

Input: An MKC (w, B, W) over I with uniform capacities. A submodular function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$.

- 1 Find an approximate solution $\bar{y} \in P_B$ for $\max_{\bar{y} \in P_B} F(\bar{y})$, where P_B is the block polytope of B , and F is the multilinear extension of f .
- 2 Choose $\mu = \min \left\{ (\log |B|)^{-\frac{1}{4}}, \frac{1}{2} \right\}$.
- 3 Set $\bar{y}^0 \leftarrow (1 - 4\mu)\bar{y}$. and let G_1, \dots, G_τ be the μ -grouping of \bar{y}^0 .
- 4 **for** $k = 1, 2, \dots, \tau$ **do** $\bar{y}^k \leftarrow \text{Pipage}(\bar{y}^{k-1}, f, G_k, \bar{1})$.
- 5 $\bar{y}' = \text{Pipage}(\bar{y}^\tau, f, L_{B,\mu}, (w(i))_{i \in L_{B,\mu}})$.
- 6 Let $S = \{i \in I \mid \bar{y}'_i = 1\}$.
- 7 Pack the items in S into B using a bin packing algorithm. Return the resulting assignment.

Proof. Let A^* be an optimal solution for the input instance, and $\text{OPT} = f(\bigcup_{b \in B} A^*(b))$ its value. By Lemma 8, $\mathbb{1}_{\bigcup_{b \in B} A^*(b)} \in P_B$. Let $c = 1 - e^{-1}$. By using the algorithm of [7] we have that $F(\bar{y}) \geq \left(c - \frac{1}{|I|}\right) \cdot \text{OPT}$ (\bar{y} is defined in Step 1 of Algorithm 1). The algorithm of [7] is used with the FPTAS of Lemma 7 as an oracle for solving linear optimization problems over P_B . We note that a $\left(c - \frac{1}{|I|}\right)$ -approximate solution can be obtained even when the algorithm is only given an FPTAS (and not an exact solver) for linear optimization problems over the polytope.

Since the multilinear extension has negative second derivatives [7], it follows that $F(\bar{y}^0) \geq (1 - 4\mu) \cdot \left(c - \frac{1}{|I|}\right) \cdot \text{OPT}$. Now, consider the vector \bar{y}' output in Step 5 of the algorithm. By Lemma 11, it follows that $\mathbb{E}[F(\bar{y}')] \geq F(\bar{y}^0) \geq (1 - 4\mu) \cdot \left(c - \frac{1}{|I|}\right) \cdot \text{OPT}$, and $\bar{y}' \in \{0, 1\}^I$ (note that $\bar{y}'_i = \bar{y}_i = 0$ for any i with $w(i) > W_B^*$ due to (1)). Thus, for the set S defined in Step 6 of the algorithm, we have $\mathbb{E}[f(S)] \geq (1 - 4\mu) \cdot \left(c - \frac{1}{|I|}\right) \cdot \text{OPT} \geq \left(c - O\left((\log |B|)^{-\frac{1}{4}}\right)\right) \cdot \text{OPT}$ (observe we may assume w.l.o.g that $|I| \geq |B|$).

To complete the proof, it remains to show that the bin packing algorithm in Step 7 packs all items in S into the bins B . By Lemma 11, for any $1 \leq k \leq \tau$, it holds that $|S \cap G_k| = \sum_{i \in G_k} \bar{y}'_i \leq 1 + \sum_{i \in G_k} \bar{y}_i^0 \leq \mu \cdot |B| + 2$ (the last inequality follows from Lemma 10). Similarly, there is $i^* \in L_{B,\mu}$ such that

$$w(S \cap L_{B,\mu}) = \sum_{i \in L_{B,\mu}} \bar{y}'_i \cdot w(i) \leq w(i^*) + \sum_{i \in L_{B,\mu}} \bar{y}_i^0 \cdot w(i) \leq \mu \cdot W_B^* + \sum_{i \in L_{B,\mu}} \bar{y}_i^0 \cdot w(i).$$

To meet the conditions of Lemma 10, we need to remove (up to) two items from each group, i.e., $S \cap G_k$, for $1 \leq k \leq \tau$. Let $R \subseteq S$ be a minimal subset such that $|(S \setminus R) \cap G_k| \leq \mu|B|$ for all $1 \leq k \leq \tau$. By the above we have that $|R| \leq 2 \cdot \tau \leq 2 \cdot (\mu^{-2} + 1)$. Therefore, $S \setminus R$ satisfies the conditions of Lemma 10. Hence, by taking $\delta = 4\mu$ and $\lambda = \mu$, the items in $S \setminus R$ can be packed into $(1 - \mu)|B| + 4 \cdot 4^{\mu^{-2}} + 2\mu$ bins. By using an additional bin for each item in R , and assuming $|B|$ is large enough, the items in S can be packed into

$$(1 - \mu)|B| + 4 \cdot 4^{\mu^{-2}} + 2\mu + 2 \cdot (\mu^{-2} + 1) \leq |B| - \frac{|B|}{(\log |B|)^{\frac{1}{4}}} + 5 \cdot 4^{\sqrt{\log |B|}} + 3 \leq |B|$$

bins of capacity W_B^* . Recall that the algorithm of [25] returns a packing in at most $\text{OPT} + O(\log^2 \text{OPT})$ bins. Thus, for large enough $|B|$, the number of bins used in Step 7 of

Algorithm 1 is at most

$$|B| - \frac{|B|}{(\log |B|)^{\frac{1}{4}}} + 5 \cdot 4^{\sqrt{\log |B|}} + O(\log^2 |B|) \leq |B|.$$

Finally, we note that Algorithm 1 can be implemented in polynomial time. \blacktriangleleft

4 Approximation Algorithm

In this section we present our algorithm for general instances of d -MKCP, which gives the result in Theorem 1. In designing the algorithm, a key observation is that we can restrict our attention to d -MKCP instances of certain structure, with other crucial properties satisfied by the objective function. For the *structure*, we assume the bins are partitioned into *levels* by capacities, using the following definition of [15].

► **Definition 13.** For any $N \in \mathbb{N}$, a set of bins B and capacities $W : B \rightarrow \mathbb{R}_{\geq 0}$, a partition $(K_j)_{j=0}^\ell$ of B is N -leveled if, for all $0 \leq j \leq \ell$, K_j is a block and $|K_j| = N^{\lfloor \frac{j}{N^2} \rfloor}$. We say that B and W are N -leveled if such a partition exists.

For $N, \xi \in \mathbb{N}$, (N, ξ) -restricted d -MKCP is the special case of d -MKCP in which for any instance $\mathcal{R} = (I, (w_t, B_t, W_t)_{t=1}^d, \mathcal{I}, f)$ it holds that B_t and W_t are N -leveled for all $1 \leq t \leq d$, and $f(\{i\}) - f(\emptyset) \leq \frac{\text{OPT}}{\xi}$ for any $i \in I$, where OPT is the value of an optimal solution for the instance. We assume the input for (N, ξ) -restricted d -MKCP includes the N -leveled partition $(K_j^t)_{j=0}^{\ell_t}$ of B_t for all $1 \leq t \leq d$. Combining standard enumeration with the structuring technique of [15], we derive the next result.

► **Lemma 14.** For any $N, \xi, d \in \mathbb{N}$ and $c \in [0, 1]$, a polynomial time c -approximation for modular/ monotone/ non-monotone (N, ξ) -restricted d -MKCP with a matroid/ matroid intersection/ matching/ no additional constraint implies a polynomial time $c \cdot (1 - \frac{d}{N})$ -approximation for d -MKCP, with the same type of function and same type of additional constraint.

The proof of the lemma is given in [16].

Our algorithm for (N, ξ) -restricted d -MKCP associates a polytope with each instance. To this end, we first generalize the definition of a block polytope (Definition 6) to represent an MKC. We then use it to define a polytope for the whole instance.

► **Definition 15.** For $\gamma > 0$, the extended γ -partition polytope of an MKC (w, B, W) and the partition $(K_j)_{j=0}^\ell$ of B to blocks is

$$P^e = \left\{ (\bar{x}, \bar{y}^0, \dots, \bar{y}^\ell) \left| \begin{array}{ll} \bar{x} \in [0, 1]^I & \\ \sum_{j=0}^\ell \bar{y}^j = \bar{x} & \\ \bar{y}^j \in P_{K_j} & \forall 0 \leq j \leq \ell \\ \bar{y}_i^j = 0 & \forall 0 \leq j \leq \ell, |K_j| = 1, i \in I \setminus L_{K_j, \gamma} \end{array} \right. \right\} \quad (7)$$

where P_{K_j} is the block polytope of K_j , and $L_{K_j, \gamma}$ is the set of γ -light items of K_j . The γ -partition polytope of (w, B, W) and $(K_j)_{j=0}^\ell$ is

$$P = \{ \bar{x} \in [0, 1]^I \mid \exists \bar{y}^0, \dots, \bar{y}^\ell \in [0, 1]^I \text{ s.t. } (\bar{x}, \bar{y}^0, \dots, \bar{y}^\ell) \in P^e \} \quad (8)$$

The last constraint in (7) forbids the assignment of γ -heavy items to blocks of a single bin. This technical requirement is used to show a concentration bound.

Finally, the γ -instance polytope of $(I, (w_t, B_t, W_t)_{t=1}^d, \mathcal{I}, f)$ and a partition $(K_j^t)_{j=0}^{\ell_t}$ of B_t to blocks, for $1 \leq t \leq d$, is $P = P(\mathcal{I}) \cap \left(\bigcap_{t=1}^d P_t\right)$, where $P(\mathcal{I})$ is the convex hull of \mathcal{I} and P_t is the γ -partition polytope of (w_t, B_t, W_t) and $(K_j^t)_{j=0}^{\ell_t}$. In the *instance polytope optimization problem*, we are given a d -MKCP instance \mathcal{R} with a partition of the bins to blocks for each MKC, $\bar{c} \in \mathbb{R}^I$ and $\gamma > 0$. The objective is to find $\bar{x} \in P$ such that $\bar{x} \cdot \bar{c}$ is maximized, where P is the γ -instance polytope of \mathcal{R} . While the problem cannot be solved exactly, it admits an FPTAS.

► **Lemma 16.** *There is an FPTAS for the instance polytope optimization problem.*

The lemma follows from known techniques for approximating an exponential size linear program using an approximate separation oracle for the dual program. The full proof appears in [16].

The next lemma asserts that the γ -instance polytope provides an approximate representation for the instance as a polytope.

► **Lemma 17.** *Given an (N, ξ) -restricted d -MKCP instance \mathcal{R} with objective function f , let $S, (A_t)_{t=1}^d$ be an optimal solution for \mathcal{R} and $\gamma > 0$. Then there is $S' \subseteq S$ such that $\mathbb{1}_{S'} \in P$ and $f(S') \geq \left(1 - \frac{N^2 \cdot d}{\xi \cdot \gamma}\right) f(S)$, where P is the γ -instance polytope of \mathcal{R} .*

Lemma 17 is proved constructively by removing the γ -heavy items assigned to blocks of a single bin in A_t , for $1 \leq t \leq d$. The full proof appears in [16].

Recall that F is the multilinear extension of the objective function f . Our algorithm finds a vector \bar{x} in the instance polytope for which $F(\bar{x})$ approximates the optimum. The fractional solution \bar{x} is then rounded to an integral solution. Initially, a random set $R \in \mathcal{I}$ is sampled, with $\Pr(i \in R) = (1 - \delta)^2 \bar{x}_i$.⁹ The technique by which R is sampled depends on \mathcal{I} . If $\mathcal{I} = 2^I$ then R is sampled according to \bar{x} , i.e., $R \sim (1 - \delta)^2 \bar{x}$ (as defined in Section 1.3). If \mathcal{I} is a matroid constraint, the sampling of [9] is used. Finally, if \mathcal{I} is a matroid intersection, or a matching constraint, then the dependent rounding technique of [10] is used. Each of the distributions admits a Chernoff-like concentration bound. These bounds are central to our proof of correctness. We refer to the above operation as sampling R by \bar{x} , δ and \mathcal{I} .

Given the set R , the algorithm proceeds to a *purging* step. While this step does not affect the content of R if f is monotone, it is critical in the non-monotone case. Given a submodular function $f : 2^I \rightarrow \mathbb{R}$, we define a purging function $\eta_f : 2^I \rightarrow 2^I$ as follows. Fix an arbitrary order over I (which is independent of S), initialize $J = \emptyset$ and iterate over the items in S by their order in I . For an item $i \in S$, if $f(J \cup \{i\}) - f(J) \geq 0$ then $J \leftarrow J \cup \{i\}$; else, continue to the next item. Now, $\eta_f(S) = J$, where J is the set at the end of the process. The purging function was introduced in [11] and is used here similarly in conjunction with the FKG inequality.

While the above sampling and purging steps can be used to select a set of items for the solution, they do not determine how these items are assigned to the bins. We now show that it suffices to associate the selected items with blocks and then use a Bin Packing algorithm for finding their assignment to the bins in the blocks, as in Algorithm 1.

Intuitively, we would like to associate a subset of items I_j^t with a block K_j^t in a way that enables to assign the items in $I_j^t \cap R$ to $|K_j^t|$ bins, for $1 \leq t \leq d$ and $1 \leq j \leq \ell_t$. Consider two cases. If $|K_j^t| > 1$ then we ensure $I_j^t \cap R$ satisfies conditions that allow using Fractional

⁹ Recall that \mathcal{I} is the additional constraint.

Grouping (see Lemma 10). On the other hand, if $|K_j^t| = 1$, it suffices to require that $R \cap I_j^t$ adheres to the capacity constraint of this bin. Such a partition $(I_j^t)_{j=0}^{\ell_t}$ of $\text{supp}(\bar{x})$ can be computed for each of the MKCs. We refer to this partition as the *Block Association* of a point in the γ -partition polytope and μ , on which the partition depends. The formal definition of block association and its properties can be found in [16].

We proceed to analyze our algorithm (see the pseudocode in Algorithm 2).

■ **Algorithm 2** (N, ξ) -restricted d -MKCP.

Input: An (N, ξ) -restricted d -MKCP instance \mathcal{R} defined by

$\left(I, (w_t, B_t, W_t)_{t=1}^d, \mathcal{I}, f \right)$ and $(K_j^t)_{j=0}^{\ell_t}$, the N -leveled partition of B_t for $1 \leq t \leq d$.

Configuration: $\gamma > 0$, $\delta > 0$, $N \in \mathbb{N}$, $\xi \in \mathbb{N}$,

- 1 Optimize $F(\bar{x})$ with $\bar{x} \in P$, where P is the γ -instance polytope of \mathcal{R} , and F is the multilinear extension of f .
 - 2 Let R be a random set sampled by \bar{x} , δ and \mathcal{I} . Define $J = \eta_f(R)$ (η_f is the purging function).
 - 3 Let $\bar{y}^{t,0}, \dots, \bar{y}^{t,\ell_t} \in [0, 1]^I$ such that $(\bar{x}, \bar{y}^{t,0}, \dots, \bar{y}^{t,\ell_t}) \in P_t^e$, where P_t^e is the extended γ -partition polytope of (w_t, B_t, W_t) and the partition $(K_j^t)_{j=0}^{\ell_t}$, for $1 \leq t \leq d$.
 - 4 Find the block association $(I_j^t)_{j=0}^{\ell_t}$ of $(1 - \delta)(\bar{x}, \bar{y}^{t,0}, \dots, \bar{y}^{t,\ell_t})$ and $\mu = \frac{\delta}{4}$ for $1 \leq t \leq d$.
 - 5 Pack the items of $J \cap I_j^t$ into the bins of K_j^t using an algorithm for bin packing if $|K_j^t| > 1$, or simply assign $J \cap I_j^t$ to K_j^t otherwise.
 - 6 Return J and the resulting assignment if the previous step succeeded; otherwise, return an empty set and an empty packing.
-

► **Lemma 18.** *For any $d \in \mathbb{N}$, $\varepsilon > 0$ and $M > 0$, there are parameters $N \in \mathbb{N}$ satisfying $N > M$, $\xi \in \mathbb{N}$, $\gamma > 0$ and $\delta > 0$ such that Algorithm 2 is a randomized $(c - \varepsilon)$ -approximation for (N, ξ) -restricted d -MKCP, where $c = 1$ for modular instances with any type of additional constraint, $c = 1 - e^{-1}$ for monotone instances with a matroid constraint, and $c = 0.385$ for non-monotone instances with no additional constraint.*

A formal proof of the lemma appears in [16]. Theorem 1 follows immediately from Lemmas 18 and 14.

References

- 1 Alexander A Ageev and Maxim I Sviridenko. Pipeage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.
- 2 Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage Knapsack. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, pages 22:1–22:14, 2019.
- 3 Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1561–1579, 2016.
- 4 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

- 5 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1433–1452. SIAM, 2014.
- 6 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, pages 182–196, 2007.
- 7 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.
- 8 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.
- 9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, page 575–584, 2010.
- 10 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 1080–1097, 2011.
- 11 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.
- 12 Reuven Cohen and Guy Grebla. Efficient allocation of periodic feedback channels in broadband wireless networks. *IEEE/ACM Transactions on Networking*, 23(2):426–436, 2014.
- 13 W Fernandez De La Vega and George S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
- 14 Yaron Fairstein, Ariel Kulik, Joseph Naor, and Danny Raz. General knapsack problems in a dynamic setting. To appear in APPROX, 2021.
- 15 Yaron Fairstein, Ariel Kulik, Joseph (Seffi) Naor, Danny Raz, and Hadas Shachnai. A $(1 - e^{-1} - \epsilon)$ -approximation for the monotone submodular multiple knapsack problem. In *28th Annual European Symposium on Algorithms (ESA 2020)*, pages 44:1–44:19, 2020.
- 16 Yaron Fairstein, Ariel Kulik, and Hadas Shachnai. Modular and submodular optimization with multiple knapsack constraints via fractional grouping, 2020. arXiv:2007.10470.
- 17 Yaron Fairstein, Ariel Kulik, and Hadas Shachnai. Tight approximations for modular and submodular optimization with d -resource multiple knapsack constraints, 2020. (second version). arXiv:2007.10470v2.
- 18 U. Feige and M. Goemans. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems, ISTCS '95*, pages 182–189, 1995.
- 19 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
- 20 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 570–579, 2011.
- 21 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum separable assignment problems. *Mathematics of Operations Research*, 36(3):416–431, 2011.
- 22 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011.
- 23 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 39(4):1392–1412, 2010.
- 24 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *SOFSEM'12*, pages 313–324, 2012. doi:10.1007/978-3-642-27660-6_26.

- 25 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 (sfcs 1982)*, pages 312–320, 1982.
- 26 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.
- 27 Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.
- 28 Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. *Information Processing Letters*, 110(16):707–710, 2010.
- 29 Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Math. Oper. Res.*, 38(4):729–739, 2013. doi:10.1287/moor.2013.0592.
- 30 Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, 2010.
- 31 S. Martello and P. Toth. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimization*, 1990.
- 32 G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.
- 33 Thomas Rothvoß. Approximating bin packing within $o(\log \text{opt}^* \log \log \text{opt})$ bins. In *IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 20–29, 2013.
- 34 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.
- 35 Xiaoming Sun, Jialin Zhang, and Zhijie Zhang. Tight algorithms for the submodular multiple knapsack problem. *arXiv preprint arXiv:2003.11450*, 2020.
- 36 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.
- 37 Deepak S Turaga, Krishna Ratakonda, and Junwen Lai. QoS support for media broadcast in a services oriented architecture. In *2006 IEEE International Conference on Services Computing (SCC'06)*, pages 127–134, 2006.
- 38 Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. Performance and power management for cloud infrastructures. In *IEEE 3rd international Conference on Cloud Computing (CLOUD)*, pages 329–336. IEEE, 2010.
- 39 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.
- 40 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.
- 41 Quanqing Xu, Khin Mi Mi Aung, Yongqing Zhu, and Khai Leong Yong. A blockchain-based storage system for data analytics in the internet of things. In *New Advances in the Internet of Things*, pages 119–138. Springer, 2018.

Differentially Private Algorithms for Graphs Under Continual Observation

Hendrik Fichtenberger ✉ 

Universität Wien, Austria

Monika Henzinger ✉

Universität Wien, Austria

Wolfgang Ost ✉

Universität Wien, Austria

Abstract

Differentially private algorithms protect individuals in data analysis scenarios by ensuring that there is only a weak correlation between the existence of the user in the data and the result of the analysis. Dynamic graph algorithms maintain the solution to a problem (e.g., a matching) on an evolving input, i.e., a graph where nodes or edges are inserted or deleted over time. They output the value of the solution after each update operation, i.e., continuously. We study (event-level and user-level) differentially private algorithms for graph problems under continual observation, i.e., differentially private dynamic graph algorithms. We present event-level private algorithms for partially dynamic counting-based problems such as triangle count that improve the additive error by a polynomial factor (in the length T of the update sequence) on the state of the art, resulting in the first algorithms with additive error polylogarithmic in T .

We also give ε -differentially private and partially dynamic algorithms for minimum spanning tree, minimum cut, densest subgraph, and maximum matching. The additive error of our improved MST algorithm is $O(W \log^{3/2} T / \varepsilon)$, where W is the maximum weight of any edge, which, as we show, is tight up to a $(\sqrt{\log T} / \varepsilon)$ -factor. For the other problems, we present a partially-dynamic algorithm with multiplicative error $(1 + \beta)$ for any constant $\beta > 0$ and additive error $O(W \log(nW) \log(T) / (\varepsilon \beta))$. Finally, we show that the additive error for a broad class of dynamic graph algorithms with user-level privacy must be linear in the value of the output solution's range.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Security and privacy → Pseudonymity, anonymity and untraceability

Keywords and phrases differential privacy, continual observation, dynamic graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.42

Related Version Full Version: <https://arxiv.org/abs/2106.14756>

Funding Research supported by the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N. Wolfgang Ost gratefully acknowledges the financial support from the Vienna Graduate School on Computational Optimization which is funded by the Austrian Science Fund (FWF project no. W1260-N35).

1 Introduction

Differential privacy aims to protect individuals whose data becomes part of an increasing number of data sets and is subject to analysis. A differentially private algorithm guarantees that its output depends only very little on an individual's contribution to the input data. Roughly speaking, an algorithm is ϵ -differentially private if the probability that it outputs O on data set D is at most an e^ϵ -factor of the probability that it outputs O on any adjacent data set D' . Two data sets are *adjacent* if they differ only in the data of a single user. Differential privacy was introduced in the setting of databases [9, 11], where users (entities) are typically represented by rows and data is recorded in columns. An important notion



© Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 42; pp. 42:1–42:16

Leibniz International Proceedings in Informatics



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

that allowed for the development of generic techniques and tools (like the Laplace and the exponential mechanism) is the *sensitivity* of a function f : the static sensitivity ρ of f is the maximum $|f(D) - f(D')|$ over all adjacent pairs D, D' . Differential privacy was later generalized to a more challenging setting, where data evolves over time [12, 6]: a differentially private algorithm under *continual observation* must provide the same privacy guarantees as before, but for a sequence (or stream) of data sets instead of just a single data set. Often, this sequence results from updates to the original data set that arrive over time. In this setting, the presence or absence of a single user in one update can affect the algorithm's output on all future data sets, i.e., two adjacent databases can differ on all future outputs and, thus, have infinite sensitivity.

In this paper, we study differentially private graph algorithms under continual observation, i.e., for *dynamic* graph problems. The input is a sequence of graphs that results from node or edge updates, i.e., insertions or deletions. *Partially dynamic* algorithms only allow either insertions or deletions, *fully dynamic* algorithms allow both. After each update, the algorithm has to output a solution for the current input, i.e., the algorithm outputs a sequence of answers that is equally long as the input sequence. For differentially private graph algorithms two notions of *adjacency of graph sequences* exist: node-adjacency and edge-adjacency. Two graph sequences are *edge-adjacent* if they only differ in a single insertion or deletion of an edge. Similarly, two graph sequences are *node-adjacent* if they only differ in an insertion or deletion of a node.¹

We initiate the study of differentially private algorithms for *non-local* partially dynamic graph problems. We consider a problem *non-local* if its (optimum) value cannot be derived from the graph's frequency histogram of constant-size subgraphs and call it *local* otherwise. Non-local problems include the cost of the minimum spanning tree, the weight of the global and s - t minimum cut, and the density of the densest subgraph. We also give improved algorithms for local graph problems and show various lower bounds on the additive error for differentially private dynamic graph algorithms.

Local problems. The only prior work on differentially private dynamic algorithms is an algorithm by Song et al. [29] for various *local* graph problems such as counting high-degree nodes, triangles and other constant-size subgraphs. We present an algorithm for these local problems that improves the additive error by a factor of $\sqrt{T}/\log^{3/2} T$, where T is the length of the update sequence. We also give the first differentially private partially-dynamic algorithm for the value of the minimum spanning tree. Table 1 lists upper and lower bounds for these results, where n is the number of nodes in the graph, W is the maximum edge weight (if applicable), D is the maximum node degree, ϵ is an arbitrarily small positive constant, and δ is the failure probability of the algorithm. We state below our main contributions in more detail. The update time of all our algorithms is linear in $\log T$ plus the time needed to solve the corresponding non-differentially private dynamic graph problem.

► **Theorem 1** (see Section 3). *Let $\epsilon, \delta > 0$. There exist an ϵ -edge-differentially as well as an ϵ -node-differentially private algorithm for partially-dynamic minimum spanning tree, edge count, the number of high-degree nodes, the degree histogram, triangle count and k -star count that with probability at least $1 - \delta$ give an answer with additive error as shown in Table 1.*

¹ Of course, a graph can also be represented by a database, where, e.g., every row corresponds to an edge, but as we present algorithms that solve graph algorithmic problems we use the graph-based terminology through the paper.

■ **Table 1** Additive errors for partially-dynamic ε -differentially private algorithms with failure probability δ . We use D for the maximum degree and W for the maximum edge weight, n for the maximum number of nodes of any graph in the input sequence, and $\Lambda = \log(1/\delta)/\varepsilon$. The upper bounds follow from Corollary 13 and Table 3 on page 11. See Section 5 for results on event-level lower bounds and Section 6 for user-level lower bounds.

Graph function	partially dynamic		fully dynamic	
	edge-adj. event-level	node-adj. event-level	edge-adj. event-level	edge-adj. user-level
min. spanning tree	$\Omega(W \log T),$ $O(W \log^{3/2} T \cdot \Lambda)$	$\Omega(W \log T),$ $O(DW \log^{3/2} T \cdot \Lambda)$	$\Omega(W \log T)$	$\Omega(nW)$
min. cut, max. matching	$\Omega(W \log T)$	$\Omega(W \log T)$	$\Omega(W \log T)$	$\Omega(nW)$
edge count	$\Omega(\log T),$ $O(\log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(D \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(\log^{3/2} T \cdot \Lambda)$	$\Omega(n^2)$
high-degree nodes	$\Omega(\log T),$ $O(\log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(D \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T)$	$\Omega(n)$
degree histogram	$\Omega(\log T),$ $O(D \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(D^2 \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T)$	$\Omega(n)$
triangle count	$\Omega(\log T),$ $O(D \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(D^2 \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T)$	$\Omega(n^3)$
k -star count	$\Omega(\log T),$ $O(D^k \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T),$ $O(D^k \log^{3/2} T \cdot \Lambda)$	$\Omega(\log T)$	$\Omega(n^{k+1})$

Non-local problems. For non-local problems we present an algorithm that, by allowing a small multiplicative error, can obtain differentially private partially dynamic algorithms for a broad class of problems that includes the aforementioned problems. Table 2 lists our results for some common graph problems. The algorithm achieves the following performance.

► **Theorem 2** (see Theorem 16). *Let $\varepsilon, \beta, \delta, r > 0$ and let f be a function with range $[1, r]$ that is monotone on all input sequences and has sensitivity ρ . There exists an ε -differentially private dynamic algorithm with multiplicative error $(1 + \beta)$, additive error $O(\rho \log(r) \log(T) / \log(1 + \delta))$ and failure probability δ that computes f .*

Note that for partially dynamic graph algorithms it holds that $T = O(n^2)$. Thus for local problems the bounds presented in Table 1 are superior to the bounds in Table 2.

Lower bounds. We complement these upper bounds by also giving some lower bounds on the additive error of any differentially private dynamic graph algorithm. For the problems in Table 1 we show lower bounds of $\Omega(W \log T)$, resp. $\Omega(\log T)$. Note that these lower bounds apply to the partially dynamic as well as to the fully dynamic setting.

The above notion of differential privacy is also known as *event-level* differential privacy, where two graph sequences differ in at most one “event”, i.e., one update operation. A more challenging notion is *user-level* differential privacy. Two graph sequences are edge-adjacent *on user-level* if they differ in *any* number of updates for a single edge (as opposed to *one* update for a single edge in the case of the former *event-level* adjacency). Note that requiring

■ **Table 2** Private algorithms with failure probability δ with additional multiplicative error of $(1 + \beta)$ for arbitrary $\beta > 0$. We use $\Lambda = 1/(\varepsilon\delta \log(1 + \beta))$, D for the maximum degree, W for the maximum edge weight and n for the maximum number of nodes of any graph in the input sequence.

Graph function	partially dynamic, event-level	
	edge-adjacency	node-adjacency
minimum cut	$O(W \log(nW) \log(T) \cdot \Lambda)$	$O(DW \log(nW) \log(T) \cdot \Lambda)$
densest subgraph	$O(\log(n) \log(T) \cdot \Lambda)$	$O(\log(n) \log(T) \cdot \Lambda)$
minimum s, t -cut	$O(W \log(nW) \log(T) \cdot \Lambda)$	$O(DW \log(nW) \log(T) \cdot \Lambda)$
maximum matching	$O(W \log(nW) \log(T) \cdot \Lambda)$	$O(W \log(nW) \log(T) \cdot \Lambda)$

user-level edge-differential privacy is a more stringent requirement on the algorithm than event-level edge-differential privacy.² We show strong lower bounds for edge-differentially private algorithms on user-level for a broad class of dynamic graph problems.

► **Theorem 3** (informal, see Theorem 19). *Let f be a function on graphs, and let G_1, G_2 be arbitrary graphs. There exists a $T \geq 1$ so that any ε -edge-differentially private dynamic algorithm on user-level that computes f must have additive error $\Omega(|f(G_1) - f(G_2)|)$ on input sequences of length at least T .*

This theorem leads to the lower bounds for fully dynamic algorithms stated in Table 1.

Technical contribution. Local problems. Our algorithms for local problems (Theorem 1) incorporate a counting scheme by Chan et al. [6] and the *difference sequence technique* by Song et al. [29]. The difference sequence technique addresses the problem that two adjacent graphs might differ on all outputs starting from the point in the update sequence where their inputs differ. More formally, let $f_{\mathcal{G}}(t)$ be the output of the algorithm after operation t in the graph sequence \mathcal{G} . Then the *continuous global sensitivity* $\sum_{t=1}^T |f_{\mathcal{G}}(t) - f_{\mathcal{G}'}(t)|$ might be $\Theta(\rho T)$. Using the “standard” Laplacian mechanism for such a large sensitivity would, thus, lead to an additive error linear in T . The idea of [29] is to use instead the *difference sequence of f* defined as $\Delta f(t) = f(t) - f(t-1)$, as they observed that for various local graph properties the continuous global sensitivity of the difference sequence, i.e., $\sum_{t=1}^T |\Delta f_{\mathcal{G}}(t) - \Delta f_{\mathcal{G}'}(t)|$ can be bounded by a function independent of T . However, their resulting partially-dynamic algorithms still have an additive error linear in \sqrt{T} . We show how to combine the continuous global sensitivity of the difference sequence with the binary counting scheme of Chan et al. [6] to achieve partially-dynamic algorithms with additive error linear in $\log^{3/2} T$.

Furthermore we show that the approach based on the continuous global sensitivity of the difference sequence fails, if the presence or absence of a node or edge can significantly change the target function’s value for all of the subsequent graphs. In particular, we show that for several graph problems like minimum cut and maximum matching changes in the function value between adjacent graph sequences can occur at every time step even for partially dynamic sequences, resulting in a continuous global sensitivity of the difference sequence that is linear in T . This implies that this technique cannot be used to achieve differentially private dynamic algorithms for these problems.

Non-local problems. We leverage the fact that the sparse vector technique [13] provides negative answers to threshold queries with little effect on the additive error to approximate monotone functions f on graphs under continual observation (e.g., the minimum cut value

² Node-adjacency on user-level is defined accordingly but not studied in this paper.

in an incremental graph) with multiplicative error $(1 + \beta)$: If r is the maximum value of f , we choose thresholds $(1 + \beta), \dots, (1 + \beta)^{\log_{1+\beta}(r)}$ for the queries. This results in at most $\log_{1+\beta}(r)$ positive answers, which affect the additive error linearly, while the at most T negative answers affect the additive error only logarithmically instead of linearly.

Lower bounds. Dwork et al. [12] had given a lower bound for counting in binary streams. We reduce this problem to partially dynamic graph problems on the event-level to achieve the event-level lower bounds.

For the user-level lower bounds we assume by contradiction that an ϵ -differentially private dynamic algorithm \mathcal{A} with “small” additive error exists and construct an exponential number of graph sequences that are all user-level “edge-close” to a simple graph sequence \mathcal{G}' . Furthermore any two such graph sequences have at least one position with two very different graphs such that \mathcal{A} (due to its small additive error) must return two different outputs at this position, which leads to two different output sequences if \mathcal{A} answers within its error bound. Let O_i be the set of accurate output sequences of \mathcal{A} on one of the graph sequences G_i . By the previous condition $O_i \cap O_j = \emptyset$ if $i \neq j$. As G_i is “edge-close” to \mathcal{G}' , there is a relatively large probability (depending on the degree of “closeness”) that O_i is output when \mathcal{A} runs on \mathcal{G}' . This holds for all i . However, since $O_i \cap O_j = \emptyset$ if $i \neq j$ and we have constructed exponentially many graph sequences G_i , the sum of these probabilities over all i adds up to a value larger than 1, which gives a contradiction. The proof is based on ideas of a lower bound proof for databases in [12].

All missing proofs can be found in the full version of the paper at <http://arxiv.org/abs/2106.14756>.

Related Work. Differential privacy, developed in [9, 11], is the de facto gold standard of privacy definitions and several lines of research have since been investigated [2, 25, 20, 10, 16, 4]. In particular, differentially private algorithms for the release of various graph statistics such as subgraph counts [19, 3, 7, 21, 32], degree distributions [18, 8, 33], minimum spanning tree [26], spectral properties [31, 1], cut problems [16, 1, 14], and parameter estimation for special classes of graphs [23] have been proposed. Dwork et al. [12] and Chan et al. [6] extended the analysis of differentially private algorithms to the regime of continual observation, i.e., to input that evolves over time. Since many data sets in applications are evolving data sets, this has lead to results for several problems motivated by practice [5, 22, 27, 15, 30]. Only one prior work analyzes evolving graphs: Song et al. [29] study problems in incremental bounded-degree graphs that are functions of local neighborhoods. Our results improve all bounds for undirected graphs initially established in [29] by a factor of $\sqrt{T}/\log^{3/2} T$ in the additive error.

2 Preliminaries

2.1 Graphs and Graph Sequences

We consider undirected graphs $G = (V, E)$, which change dynamically. Graphs may be edge-weighted, in which case $G = (V, E, w)$, where $w : E \rightarrow \mathbb{N}$. The evolution of a graph is described by a *graph sequence* $\mathcal{G} = (G_1, G_2, \dots)$, where $G_t = (V_t, E_t)$ is derived from G_{t-1} by applying updates, i.e., inserting or deleting nodes or edges. We denote by $|\mathcal{G}|$ the length of \mathcal{G} , i.e., the number of graphs in the sequence. At time t we delete a set of nodes ∂V_t^- along with the corresponding edges ∂E_t^- and insert a set of nodes ∂V_t^+ and edges ∂E_t^+ . More formally, $V_t = (V_{t-1} \setminus \partial V_t^-) \cup \partial V_t^+$ and $E_t = (E_{t-1} \setminus \partial E_t^-) \cup \partial E_t^+$, with initial node and edge sets V_0, E_0 , which may be non-empty. If a node v is deleted, then

all incident edges are deleted at the same time, i.e., if $v \in \partial V_t^-$, then $(u, v) \in \partial E_t^-$ for all $(u, v) \in E_{t-1}$. Both endpoints of an edge inserted at time t need to be in the graph at time t , i.e., $\partial E_t^+ \subseteq ((V_{t-1} \setminus \partial V_t^-) \cup \partial V_t^+) \times ((V_{t-1} \setminus \partial V_t^-) \cup \partial V_t^+)$. The tuple $(\partial V_t^+, \partial V_t^-, \partial E_t^+, \partial E_t^-)$ is the *update* at time t . For any graph G and any update u , let $G \oplus u$ be the graph that results from applying u on G .

A graph sequence is *incremental* if $\partial E_t^- = \partial V_t^- = \emptyset$ at all time steps t . A graph sequence is *decremental* if $\partial E_t^+ = \partial V_t^+ = \emptyset$ at all time steps t . Incremental and decremental graph sequences are called *partially dynamic*. Graph sequences that are neither incremental nor decremental are *fully dynamic*.

Our goal is to continually release the value of a *graph function* f which takes a graph as input and outputs a real number. In other words, given a graph sequence $\mathcal{G} = (G_1, G_2, \dots)$ we want to compute the sequence $f(\mathcal{G}) = (f(G_1), f(G_2), \dots)$. We write $f(t)$ for $f(G_t)$. Our algorithms will compute an update to the value of f at each time step, i.e., we compute $\Delta f(t) = f(t) - f(t-1)$. We call the sequence Δf the *difference sequence* of f .

Given a graph function g the *continuous global sensitivity* $\text{GS}(g)$ of g is defined as the maximum value of $\|g(S) - g(S')\|_1$ over all adjacent graph sequences S, S' . We will define adjacency of graph sequences below. In our case, we are often interested in the continuous global sensitivity of the difference sequence of a graph function f , which is given by the maximum value of $\sum_{t=1}^T |\Delta f_{\mathcal{G}}(t) - \Delta f_{\mathcal{G}'}(t)|$, where $\Delta f_{\mathcal{G}}$ and $\Delta f_{\mathcal{G}'}$ are the difference sequences of f corresponding to adjacent graph sequences \mathcal{G} and \mathcal{G}' .

Two graphs are *edge-adjacent* if they differ in one edge. We also define global sensitivity of functions applied to a single graph. Let g be a graph function. Its *static global sensitivity* $\text{GS}_{\text{static}}(g)$ is the maximum value of $|g(G) - g(G')|$ over all edge-adjacent graphs G, G' .

2.2 Differential Privacy

The range of an algorithm \mathcal{A} , $\text{Range}(\mathcal{A})$, is the set of all possible output values of \mathcal{A} . We denote the Laplace distribution with mean μ and scale b by $\text{Lap}(\mu, b)$. If $\mu = 0$, we write $\text{Lap}(b)$.

► **Definition 4** (ε -differential privacy). *A randomized algorithm \mathcal{A} is ε -differentially private if for any two adjacent databases B, B' and any $S \subseteq \text{Range}(\mathcal{A})$ we have $\Pr[\mathcal{A}(B) \in S] \leq e^\varepsilon \cdot \Pr[\mathcal{A}(B') \in S]$. The parameter ε is called the privacy loss of \mathcal{A} .*

To apply Definition 4 to graph sequences we now define adjacency for graph sequences. First, we define edge-adjacency, which is useful if the data to be protected is associated with the edges in the graph sequence. Then, we define node-adjacency, which provides stronger privacy guarantees.

► **Definition 5** (Edge-adjacency). *Let $\mathcal{G}, \mathcal{G}'$ be graph sequences as defined above with associated sequences of updates $(\partial V_t^-), (\partial V_t^+), (\partial E_t^-), (\partial E_t^+)$ and $(\partial V_t^{-'}), (\partial V_t^{+'}), (\partial E_t^{-'}), (\partial E_t^{+'})$. Let $\partial V_t^- = \partial V_t^{-'}$ and $\partial V_t^+ = \partial V_t^{+'}$ for all t . Let the initial node and edge sets for \mathcal{G} and \mathcal{G}' be $V_0 = V_0'$ and $E_0 = E_0'$. Assume w.l.o.g. that $\partial E_t^{-'} \subseteq \partial E_t^-$ and $\partial E_t^{+'} \subseteq \partial E_t^+$ for all t . The graph sequences \mathcal{G} and \mathcal{G}' are adjacent on e^* if $|\mathcal{G}| = |\mathcal{G}'|$, there exists an edge e^* and one of the following statements holds:*

1. $\partial E_t^- = \partial E_t^{-'} \forall t$ and $\exists t^*$ such that $\partial E_{t^*}^+ \setminus \partial E_{t^*}^{+'} = \{e^*\}$ and $\partial E_t^+ = \partial E_t^{+'} \forall t \neq t^*$;
2. $\partial E_t^+ = \partial E_t^{+'} \forall t$ and $\exists t^*$ such that $\partial E_{t^*}^- \setminus \partial E_{t^*}^{-'} = \{e^*\}$ and $\partial E_t^- = \partial E_t^{-'} \forall t \neq t^*$;

Remark. If \mathcal{G} and \mathcal{G}' are edge-adjacent, then for any index i the graphs at index i in the two sequences are edge-adjacent.

Two edge-adjacent graph sequences differ in either the insertion or the deletion of a single edge e^* . There are several special cases that fit into this definition. For example, we may have $\partial V_t^- = \partial V_t^{-'} = \partial V_t^+ = \partial V_t^{+'} = \emptyset$, so only edge updates would be allowed. Similarly, we can use the definition in the incremental setting by assuming $\partial V_t^- = \partial V_t^{-'} = \partial E_t^- = \partial E_t^{-'} = \emptyset$.

The definition of node-adjacency is similar to that of edge-adjacency, but poses additional constraints on the edge update sets.

► **Definition 6 (Node-adjacency).** *Let $\mathcal{G}, \mathcal{G}'$ be graph sequences as defined above with associated sequences of updates $(\partial V_t^-), (\partial V_t^+), (\partial E_t^-), (\partial E_t^+)$ and $(\partial V_t^{-'}), (\partial V_t^{+'}), (\partial E_t^{-'}), (\partial E_t^{+'})$. Assume w.l.o.g. that $\partial V_t^{-'} \subseteq \partial V_t^-$ and $\partial V_t^{+'} \subseteq \partial V_t^+$ for all t . The graph sequences \mathcal{G} and \mathcal{G}' are adjacent on v^* if $|\mathcal{G}| = |\mathcal{G}'|$, there exists a node v^* and one of the following statements holds:*

1. $\partial V_t^- = \partial V_t^{-'} \forall t$ and $\exists t^*$ such that $\partial V_t^+ \setminus \partial V_t^{+'} = \{v^*\}$ and $\partial V_t^+ = \partial V_t^{+'} \forall t \neq t^*$;
 2. $\partial V_t^+ = \partial V_t^{+'} \forall t$ and $\exists t^*$ such that $\partial V_t^- \setminus \partial V_t^{-'} = \{v^*\}$ and $\partial V_t^- = \partial V_t^{-'} \forall t \neq t^*$;
- Additionally, all edges in ∂E_t^+ and ∂E_t^- are incident to at least one node in ∂V_t^+ and ∂V_t^- , respectively. Lastly, we require that $\partial E_t^{+'} (\partial E_t^{-'})$ is the maximal subset of $\partial E_t^+ (\partial E_t^-)$ that does not contain edges incident to v^* .*

We define the following notions of differential privacy based on these definitions of adjacency.

► **Definition 7.** *An algorithm is ε -edge-differentially private (on event-level) if it is ε -differentially private when considering edge-adjacency. An algorithm is ε -node-differentially private (on event-level) if it is ε -differentially private when considering node-adjacency.*

When explicitly stated, we consider a stronger version of ε -differential privacy, which provides adjacency on user-level. While adjacency on event-level only allows two graph sequences to differ in a single update, user-level adjacency allows any number of updates to differ as long as they affect the same edge (for edge-adjacency) or node (for node-adjacency), respectively.

► **Definition 8.** *Let $\mathcal{G} = (G_1, \dots), \mathcal{G}' = (G'_1, \dots)$ be graph sequences. The two sequences are edge-adjacent on user-level if there exists an edge e^* and a sequence of graph sequences $\mathcal{S} = (\mathcal{G}_1, \dots, \mathcal{G}_\ell)$ so that $\mathcal{G}_1 = \mathcal{G}, \mathcal{G}_\ell = \mathcal{G}'$ and, for any $i \in [\ell-1]$, \mathcal{G}_i and \mathcal{G}_{i+1} are edge-adjacent on e^* . An algorithm is ε -edge-differentially private on user-level if it is ε -differentially private when considering edge-adjacency on user-level.*

2.3 Counting Mechanisms

Some of our algorithms for releasing differentially private estimates of functions on graph sequences rely on algorithms for counting in streams.

A *stream* $\sigma = \sigma(1)\sigma(2)\dots$ is a string of items $\sigma(i) \in \{L_1, \dots, L_2\} \subseteq \mathbb{Z}$, where the i -th item is associated with the i -th time step. A *binary stream* has $L_1 = 0$ and $L_2 = 1$. We denote the length of a stream, i.e., the number of time steps in the stream, by $|\sigma|$. Stream σ and σ' are adjacent if $|\sigma| = |\sigma'|$ and if there exists one and only one t^* such that $\sigma(t^*) \neq \sigma'(t^*)$ and $\sigma(t) = \sigma'(t)$ for all $t \neq t^*$.

A *counting mechanism* $\mathcal{A}(\sigma)$ takes a stream σ and outputs a real number for every time step. For all time steps t , \mathcal{A} 's output at time t is independent of all $\sigma(i)$ for $i > t$. At each time t a counting mechanism should estimate the count $c(t) = \sum_{i=1}^t \sigma(i)$.

Following Chan et al. [6] we describe our mechanisms in terms of *p-sums*, which are partial sums of the stream over a time interval. For a p-sum p we denote the beginning and end of the time interval by $\text{start}(p)$ and $\text{end}(p)$, respectively. With this notation the value of p is $\sum_{t=\text{start}(p)}^{\text{end}(p)} \sigma(t)$. To preserve privacy we add noise to p-sums and obtain *noisy p-sums*: given a p-sum p , a noisy p-sum is $\hat{p} = p + \gamma$, where γ is drawn from a Laplace-distribution.

2.4 Sparse Vector Technique

The *sparse vector technique* (SVT) was introduced by Dwork et al. [13] and was subsequently improved [17, 28]. SVT can be used to save privacy budget whenever a sequence of threshold queries f_1, \dots, f_T is evaluated on a database, but only $c \ll T$ queries are expected to exceed the threshold. Here, a threshold query asks whether a function f_i evaluates to a value above some threshold t_i on the input database. Using SVT, only queries that are answered positively reduce the privacy budget. We use the following variant of SVT, which is due to Lyu et al.

► **Lemma 9** ([24]). *Let D be a database, $\epsilon, \rho, c > 0$ and let $(f_1, t_1), \dots$ be a sequence of mappings f_i from input databases to \mathbb{R} and thresholds $t_i \in \mathbb{R}$, which may be generated adaptively one after another so that $\rho \geq \max_i \text{GS}_{\text{static}}(f_i)$. Algorithm 1 is ϵ -private.*

■ **Algorithm 1** SVT algorithm [24].

```

1 Function InitializeSvt( $D, \rho, \epsilon, c$ )
2    $\epsilon_1 \leftarrow \epsilon/2, \zeta \leftarrow \text{Lap}(\rho/\epsilon_1), \epsilon_2 \leftarrow \epsilon - \epsilon_1, \text{count} \leftarrow 0$ 
3 Function ProcessSvtQuery( $f_i, t_i$ )
4    $\nu_i \leftarrow \text{Lap}(2c\rho/\epsilon_2)$ 
5   if  $\text{count} \geq c$  then
6     return abort
7   if  $f_i(D) + \nu_i \geq t_i + \zeta$  then
8      $\text{count} \leftarrow \text{count} + 1$ , return  $\top$ 
9   else
10    return  $\perp$ 

```

3 Mechanisms Based on Continuous Global Sensitivity

Some of our mechanisms for privately estimating graph functions are based on mechanisms for counting in streams. In both settings, we compute the sum of a sequence of numbers and we will show that the mechanisms for counting can be transferred to the graph setting. However, there are differences in the analysis. In counting, the input streams differ at only one time step. This allows us to bound the difference in the true value between adjacent inputs and leads to low error. In the graph setting, the sequence of numbers can vary at many time steps. Here however, we use properties of the counting mechanisms to show that the total difference for this sequence can still be bounded, which results in the same error as in the counting setting.

We first generalize the counting mechanisms by Chan et al. [6] to streams of integers with bounded absolute value, and then transfer them to estimating graph functions.

3.1 Non-Binary Counting

We generalize the counting mechanisms of Chan et al. [6] to streams of numbers in $\{-L, \dots, L\}$, for some constant L . We view these algorithms as releasing noisy p-sums from which the count can be estimated. The generic algorithm is outlined in Algorithm 2 on page 9.

The algorithm releases a vector of noisy p-sums over T time steps, such that at every time step the noisy p-sums needed to estimate the count up to this time are available. Each of the noisy p-sums is computed exactly once. See the proof of Corollary 13 for an example on how to use p-sums.

In order to achieve the desired privacy loss the mechanisms need to meet the following requirements. Let \mathcal{A} be a counting mechanism. We define $\text{Range}(\mathcal{A}) = \mathbb{R}^k$, where k is the total number of p-sums used by \mathcal{A} and every item of the vector output by \mathcal{A} is a p-sum. We assume that the time intervals represented by the p-sums in the output of \mathcal{A} are deterministic and only depend on the length T of the input stream. For example, consider any two streams σ and σ' of length T . The ℓ -th element of $\mathcal{A}(\sigma)$ and $\mathcal{A}(\sigma')$ will be p-sums of the same time interval $[\text{start}(\ell), \text{end}(\ell)]$. We further assume that the p-sums are computed independently from each other in the following way: \mathcal{A} computes the true p-sum and then adds noise from $\text{Lap}(z \cdot \varepsilon^{-1})$, where z is a sensitivity parameter. The next lemma is stated informally in [6].

■ **Algorithm 2** Generic counting mechanism.

- 1 **Input:** privacy loss ε , stream σ of items $\{L_1, \dots, L_2\}$ with $|\sigma| = T$
 - 2 **Output:** vector of noisy p-sums $a \in \mathbb{R}^k$, released over T time steps
 - 3 **Initialization:** Determine which p-sums to compute based on T
 - 4 At each time step $t \in \{1, \dots, T\}$:
 - 5 Compute new p-sums p_i, \dots, p_j for t
 - 6 **For** $\ell = i, \dots, j$:
 - 7 $\hat{p}_\ell = p_\ell + \gamma_\ell, \quad \gamma_\ell \sim \text{Lap}((L_2 - L_1)\varepsilon^{-1})$
 - 8 Release new noisy p-sums $\hat{p}_i, \dots, \hat{p}_j$
-

► **Lemma 10** (Observation 1 from [6]). *Let \mathcal{A} be a counting mechanism as described in Algorithm 2 with $L_1 = 0$ and $L_2 = 1$ that releases k noisy p-sums, such that the count at any time step can be computed as the sum of at most y p-sums and every item is part of at most x p-sums. Then, \mathcal{A} is $(x \cdot \varepsilon)$ -differentially private, and the error is $O(\varepsilon^{-1} \sqrt{y} \log \frac{1}{\delta})$ with probability at least $1 - \delta$ at each time step.*

To extend the counting mechanisms by Chan et al. to non-binary streams of values in $\{-L, \dots, L\}$, we only need to account for the increased sensitivity in the scale of the Laplace distribution. By Lemma 11, we can use the mechanisms of Chan et al. [6] to compute the sum of a stream of numbers in $\{-L, \dots, L\}$, but gain a factor $2L$ in the error.

► **Lemma 11** (Extension of Lemma 10). *Let \mathcal{A} be a mechanism as in Algorithm 2 (see page 9) with $L_1 = -L$ and $L_2 = L$ that releases k noisy p-sums, such that the count at any time step can be computed as the sum of at most y p-sums, and every item is part of at most x p-sums. Furthermore, \mathcal{A} adds noise $\text{Lap}(2L/\varepsilon)$ to every p-sum. Then, \mathcal{A} is $(x \cdot \varepsilon)$ -differentially private, and the error is $O(L\varepsilon^{-1} \sqrt{y} \log \frac{1}{\delta})$ with probability at least $1 - \delta$ at each time step.*

Algorithm 3 Generic graph sequence mechanism.

Input: privacy loss ε , contin. global sensitivity Γ , graph sequence $\mathcal{G} = (G_1, \dots, G_T)$
Output: vector of noisy p-sums $a \in \mathbb{R}^k$, released over T time steps

- 1 **Initialization:** Determine which p-sums to compute based on T
- 2 At each time step $t \in \{1, \dots, T\}$:
 - 3 Compute $f(t)$ and $\Delta f(t) = f(t) - f(t-1)$, $f(0) := 0$
 - 4 Compute new p-sums p_i, \dots, p_j for the sequence Δf
 - 5 **For** $\ell = i, \dots, j$:
 - 6 $\hat{p}_\ell = p_\ell + \gamma_\ell$, $\gamma_\ell \sim \text{Lap}(\Gamma\varepsilon^{-1})$
 - 7 Release new noisy p-sums $\hat{p}_i, \dots, \hat{p}_j$

3.2 Graph Functions via Counting Mechanisms

We adapt the counting mechanisms to continually release graph functions by following the approach by Song et al. [29]. Algorithm 3 outlines the generic algorithm. It is similar to Algorithm 2, with the difference that the stream of numbers to be summed is computed from a graph sequence \mathcal{G} . The algorithm is independent of the notion of adjacency of graph sequences, however the additive error is linear in the continuous global sensitivity of the difference sequence Δf .

In counting binary streams we considered adjacent inputs that differ at exactly one time step. In the graph setting however, the stream of numbers that we sum, i.e., the difference sequence Δf , can differ in multiple time steps between two adjacent graph sequences. We illustrate this with a simple example. Let f be the function that counts the number of edges in a graph and consider two node-adjacent incremental graph sequences $\mathcal{G}, \mathcal{G}'$. \mathcal{G} contains an additional node v^* , that is not present in \mathcal{G}' . Whenever a neighbor is added to v^* in \mathcal{G} , the number of edges in \mathcal{G} increases by more than the number of edges in \mathcal{G}' . Thus, every time a neighbor to v^* is inserted, the difference sequence of f will differ between \mathcal{G} and \mathcal{G}' .

To generalize this, consider two adjacent graph sequences $\mathcal{G}, \mathcal{G}'$ that differ in an update at time t' . Let $\Delta f_{\mathcal{G}}$ and $\Delta f_{\mathcal{G}'}$ be the difference sequences used to compute the graph function f on \mathcal{G} and \mathcal{G}' . As discussed above we may have $\Delta f_{\mathcal{G}}(t) \neq \Delta f_{\mathcal{G}'}(t)$ for all $t \geq t'$. Thus, more than x p-sums can be different, which complicates the proof of the privacy loss. However, we observe that the set P of p-sums with differing values can be partitioned into x sets P_1, \dots, P_x , where the p-sums in each P_i cover disjoint time intervals. By using a bound on the continuous global sensitivity of the difference sequence Δf , this will lead to a privacy loss of $x \cdot \varepsilon$. The following lemma formalizes our result.

► **Lemma 12** (Lemma 11 for graph sequences). *Let f be a graph function whose difference sequence has continuous global sensitivity Γ . Let $0 < \delta < 1$ and $\varepsilon > 0$. Let \mathcal{A} be a mechanism to estimate f as in Algorithm 3 that releases k noisy p-sums and satisfies the following conditions:*

1. *at any time step the value of a graph function f can be estimated as the sum of at most y noisy p-sums,*
2. *\mathcal{A} adds independent noise from $\text{Lap}(\Gamma/\varepsilon)$ to every p-sum,*
3. *the set P of p-sums computed by the algorithm can be partitioned into at most x subsets P_1, \dots, P_x , such that in each partition P_x all p-sums cover disjoint time intervals. That is, for all $P_i \in \{P_1, \dots, P_x\}$ and all $j, k \in P_i$, $j \neq k$, it holds that (1) $\text{start}(j) \neq \text{start}(k)$ and (2) $\text{start}(j) < \text{start}(k) \implies \text{end}(j) < \text{start}(k)$.*

Then, \mathcal{A} is $(x \cdot \varepsilon)$ -differentially private, and the error is $O(\Gamma\varepsilon^{-1}\sqrt{y}\log \frac{1}{\delta})$ with probability at least $1 - \delta$ at each time step.

■ **Table 3** Global sensitivity of difference sequences.

Graph Function f	Continuous Global Sensitivity of Δf			
	Partially Dynamic		Fully Dynamic	
	node-adjacency	edge-adjacency	node-adj.	edge-adj.
edge count ^a	D	1	$\geq T$	2
high-degree nodes ^a	$2D + 1$	4	$\geq T$	$\geq T$
degree histogram ^a	$4D^2 + 2D + 1$	$8D$	$\geq 2T$	$\geq 2T$
triangle count ^a	$\binom{D}{3}$	D	$\geq T$	$\geq T$
k -star count ^a	$D \binom{D-1}{k-1} + \binom{D}{k}$	$2 \cdot \left(\binom{D}{k} - \binom{D-1}{k} \right)$	$\geq T$	$\geq T$
minimum spanning tree	$2DW$	$2W - 2$	$\geq T$	$\geq T$
minimum cut	$\geq T$	$\geq T$	$\geq T$	$\geq T$
maximum matching	$\geq T$	$\geq T$	$\geq T$	$\geq T$

^aBounds for partially dynamic node-adjacency from Song et al. [29]

We can compute the p-sums in Algorithm 3 as in the binary mechanism [6] to release ε -differentially private estimates of graph functions.

► **Corollary 13** (Binary mechanism). *Let f be a graph function whose difference sequence has continuous global sensitivity Γ . Let $0 < \delta < 1$ and $\varepsilon > 0$. For each $T \in \mathbb{N}$ there exists an ε -differentially private algorithm to estimate f on a graph sequence which has error $O(\Gamma \varepsilon^{-1} \cdot \log^{3/2} T \cdot \log \delta^{-1})$ with probability at least $1 - \delta$.*

3.3 Bounds on Continuous Global Sensitivity

Song et al. [29] give bounds on the continuous global sensitivity of the difference sequence for several graph functions in the incremental setting in terms of the maximum degree D . Table 3 summarizes the results on the continuous global sensitivity of difference sequences for a variety of problems in the partially dynamic and fully dynamic setting, both for edge- and node-adjacency. Note that bounds on the continuous global sensitivity of the difference sequence for incremental graph sequences hold equally for decremental graph sequences as for every incremental graph sequence there exists an equivalent decremental graph sequence which deletes nodes and edges in the reverse order.

In the partially dynamic setting, the continuous global sensitivity based approach works well for graph functions that can be expressed as the sum of local functions on the neighborhood of nodes. For non-local problems the approach is less successful. For the weight of a minimum spanning tree the continuous global sensitivity of the difference sequence is independent of the length of the graph sequence. However, for minimum cut and maximum matching this is not the case. In the fully-dynamic setting the approach seems not to be useful. Here, we can show that even for estimating the number of edges the continuous global sensitivity of the difference sequence scales linearly with T under node-adjacency. When considering edge-adjacency we only have low sensitivity for the edge count.

Using Corollary 13 we obtain ε -differentially private mechanisms with additive error that scales with $\log^{3/2} T$, compared to the factor \sqrt{T} in [29]. Note that we recover their algorithm when using the Simple Mechanism II by Chan et al. [6] to sum the difference sequence.

The difference sequence approach can be employed to privately estimate the weight of a minimum spanning tree in partially dynamic graph sequences. If the edge weight is bounded by W , then the continuous global sensitivity of the difference sequence is $O(W)$ and $O(DW)$ under edge-adjacency and node-adjacency, respectively. Using Corollary 13 we obtain the algorithms outlined in Theorems 14 and 15.

► **Theorem 14.** *There exists an ε -edge-differentially private algorithm that outputs the weight of a minimum spanning tree on an incremental graph sequence \mathcal{G} with edge-weights from the set $\{1, \dots, W\}$. At every time step, the algorithm has error $O(W\varepsilon^{-1} \cdot \log^{3/2} T \cdot \log \delta^{-1})$ with probability at least $1 - \delta$, where T is the length of the graph sequence.*

► **Theorem 15.** *There exists an ε -node-differentially private algorithm that outputs the weight of a minimum spanning tree on an incremental graph sequence \mathcal{G} with edge-weights from the set $\{1, \dots, W\}$. At every time step, the algorithm has error $O(DW\varepsilon^{-1} \cdot \log^{3/2} T \cdot \log \delta^{-1})$ with probability at least $1 - \delta$, where T is the length of the graph sequence and D is the maximum degree.*

4 Upper Bound for Monotone Functions

In Section 3.3, we show that privately releasing the difference sequence of a graph sequence does not lead to good error guarantees for partially dynamic problems like minimum cut. Intuitively, the reason is that even for neighboring graph sequences $\mathcal{G}, \mathcal{G}'$, the differences of the difference sequence can be non-zero for all graphs G_i, G'_i . In other words, the difference of objective values for the graphs G_i and G'_i can constantly fluctuate during continual updates. However, the difference of objective values, regardless of fluctuations, is *always small*. We show that, by allowing an arbitrarily small *multiplicative* error, we can leverage this fact for a broad class of partially dynamic problems. In particular, we prove that there exist ε -differentially private algorithms for all dynamic problems that are non-decreasing (or non-increasing) on all valid input sequences. This includes, e.g., minimum cut, maximum matching and densest subgraph on partially dynamic inputs. See Algorithm 4 for the details and Table 2 on page 4 for explicit upper bounds for applications. We state the result for monotonically increasing functions, but it is straightforward to adapt the algorithm to monotonically decreasing functions.

■ **Algorithm 4** Multiplicative error algorithm for monotone functions.

```

1 Function Initialize( $D, \rho, \epsilon, r, \beta$ )
2    $k_0 \leftarrow 0$ 
3   InitializeSvt ( $D, \rho, \epsilon, \log_{1+\beta}(r)$ )           // see Algorithm 1
4 Function Process( $f_i$ )
5    $k_i \leftarrow k_{i-1}$ 
6   while ProcessSvtQuery( $f_i, (1 + \beta)^{k_i} = \top$ ) do           // see Algorithm 1
7      $k_i \leftarrow k_i + 1$ 
8   return  $(1 + \beta)^{k_i}$ 

```

► **Theorem 16.** *Let $r > 0$ and let f be any monotonically increasing function on dynamic inputs (e.g., graphs) with range $[1, r]$ and static global sensitivity $\rho := \text{GS}_{\text{static}}(f)$. Let $\beta \in (0, 1), \delta > 0$ and let $\alpha = 16 \log_{1+\beta}(r) \rho \cdot \ln(2T/\delta)/\epsilon$. There exists an ε -differentially private algorithm for computing f with multiplicative error $(1 + \beta)$, additive error α and failure probability δ .*

5 Lower Bounds for Event-Level Privacy

We can show lower bounds for the error of edge- and node-differentially private algorithms in the partially dynamic setting. We derive the bounds by reducing differentially private counting in binary streams to these problems and apply a lower bound of Dwork et al. [12], which we restate here.

► **Theorem 17** (Lower bound for counting in binary streams [12]). *Any differentially private event-level algorithm for counting over T rounds must have error $\Omega(\log T)$ (even with $\varepsilon = 1$).*

We obtain a lower bound of $\Omega(W \log T)$ for minimum cut, maximum weighted matching and minimum spanning tree. The same approach yields a lower bound of $\Omega(\log T)$ for the subgraph-counting problems, counting the number of high-degree nodes and the degree histogram. See Table 1 on page 3. Note that any lower bound for the incremental setting can be transferred to the decremental setting, using the same reductions but proceeding in reverse.

6 Lower Bound for User-Level Privacy

We show that for several fundamental problems on dynamic graphs like minimum spanning tree and minimum cut, a differentially private algorithm with edge-adjacency on user-level must have an additive error that is linear in the maximum function value. Technically, we define the *spread* of a graph function as the maximum difference of the function's value on any two graphs. Then, we show that any algorithm must have an error that is linear in the graph function's spread. We write this section in terms of edge-adjacency but the corresponding result for node-adjacency carries over.

► **Definition 18.** *Let G_1 and G_2 be a pair of graphs. We define $\tau(G_1, G_2)$ to be an update sequence u_1, \dots, u_ℓ of minimum length that transforms G_1 into G_2 . We denote the graph sequence that results from applying $\tau(G_1, G_2)$ to G_1 by $T(G_1, G_2)$.*

Let $s, \ell : \mathbb{N} \rightarrow \{2i \mid i \in \mathbb{N}\}$ be functions. A graph function f has spread $(s(n), \ell(n))$ on inputs of size n if, for every n , there exist two graphs G_1, G_2 of size n so that $|f(G_1) - f(G_2)| \geq s(n)$ and $|\tau(G_1, G_2)| = \ell(n)$.

See Table 1 on page 3 for the resulting lower bounds.

► **Theorem 19.** *Let $\epsilon, \delta > 0$ and let f be a graph function with spread (s, ℓ) that spares an edge e . For streams of length T on graphs of size n , where $T > \log(e^{4\epsilon\ell}/(1 - \delta)) \in O(\epsilon\ell + \log(1/(1 - \delta)))$, every ϵ -differentially private dynamic algorithm with user-level edge-adjacency that computes f with probability at least $1 - \delta$ must have error $\Omega(s(n))$.*

► **Fact 20.** *Minimum spanning tree has spread $(\Theta(nW), \Theta(n))$. Minimum cut has spread $(\Theta(nW), \Theta(n^2))$. Maximal matching has spread $(\Theta(n), \Theta(n))$. Maximum cardinality matching has spread $(\Theta(n), \Theta(n))$. Maximum weight matching has spread $(\Theta(nW), \Theta(n))$.*

References

- 1 Raman Arora and Jalaj Upadhyay. On differentially private graph sparsification and applications. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 13399–13410. Curran Associates, Inc., 2019.

- 2 Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '07, pages 273–282, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1265530.1265569.
- 3 Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 87–96, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2422436.2422449.
- 4 BlumAvrim, LigettKatrina, and RothAaron. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, May 2013. doi:10.1145/2450142.2450148.
- 5 T. H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams. In Simone Fischer-Hübner and Matthew Wright, editors, *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 140–159, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31680-7_8.
- 6 T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), 2011. doi:10.1145/2043621.2043626.
- 7 Shixi Chen and Shuigeng Zhou. Recursive mechanism: Towards node differential privacy and unrestricted joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 653–664, New York, NY, USA, June 2013. Association for Computing Machinery. doi:10.1145/2463676.2465304.
- 8 Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing Graph Degree Distribution with Node Differential Privacy. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 123–138, New York, NY, USA, June 2016. Association for Computing Machinery. doi:10.1145/2882903.2926745.
- 9 Cynthia Dwork. Differential Privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 1–12, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11787006_1.
- 10 Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 371–380, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1536414.1536466.
- 11 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 265–284, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11681878_14.
- 12 Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 715–724, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806787.
- 13 Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: Efficient algorithms and hardness results. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 381–390, New York, NY, USA, May 2009. Association for Computing Machinery. doi:10.1145/1536414.1536467.
- 14 Marek Eliás, Michael Kapralov, Janardhan Kulkarni, and Yin Tat Lee. Differentially Private Release of Synthetic Graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 560–578. Society for Industrial and Applied Mathematics, December 2019. doi:10.1137/1.9781611975994.34.
- 15 M. A. Erdogdu and N. Fawaz. Privacy-utility trade-off under continual observation. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 1801–1805, June 2015. doi:10.1109/ISIT.2015.7282766.

- 16 Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially Private Combinatorial Optimization. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 1106–1125. Society for Industrial and Applied Mathematics, January 2010. doi:10.1137/1.9781611973075.90.
- 17 Moritz Hardt and Guy N. Rothblum. A Multiplicative Weights Mechanism for Privacy-Preserving Data Analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70, October 2010. doi:10.1109/FOCS.2010.85.
- 18 M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate Estimation of the Degree Distribution of Private Networks. In *2009 Ninth IEEE International Conference on Data Mining*, pages 169–178, 2009. doi:10.1109/ICDM.2009.11.
- 19 Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, August 2011. doi:10.14778/3402707.3402749.
- 20 S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What Can We Learn Privately? In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 531–540, 2008. doi:10.1109/FOCS.2008.27.
- 21 Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing Graphs with Node Differential Privacy. In Amit Sahai, editor, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 457–476, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-36594-2_26.
- 22 Georgios Kellaris, Stavros Papadopoulos, Xiaokui Xiao, and Dimitris Papadias. Differentially private event sequences over infinite streams. *Proceedings of the VLDB Endowment*, 7(12):1155–1166, 2014. doi:10.14778/2732977.2732989.
- 23 Wentian Lu and Gerome Miklau. Exponential random graph estimation under differential privacy. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 921–930, New York, NY, USA, August 2014. Association for Computing Machinery. doi:10.1145/2623330.2623683.
- 24 Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *Proceedings of the VLDB Endowment*, 10(6), 2017.
- 25 F. McSherry and K. Talwar. Mechanism Design via Differential Privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103, October 2007. doi:10.1109/FOCS.2007.66.
- 26 Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 75–84, New York, NY, USA, June 2007. Association for Computing Machinery. doi:10.1145/1250790.1250803.
- 27 J. Le Ny and G. J. Pappas. Differentially Private Filtering. *IEEE Transactions on Automatic Control*, 59(2):341–354, 2014. doi:10.1109/TAC.2013.2283096.
- 28 Aaron Roth and Tim Roughgarden. Interactive privacy via the median mechanism. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, pages 765–774, New York, NY, USA, June 2010. Association for Computing Machinery. doi:10.1145/1806689.1806794.
- 29 Shuang Song, Susan Little, Sanjay Mehta, Staal Vinterbo, and Kamalika Chaudhuri. Differentially private continual release of graph statistics, 2018. arXiv:1809.02575.
- 30 Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren. Real-Time and Spatio-Temporal Crowd-Sourced Social Network Data Publishing with Differential Privacy. *IEEE Transactions on Dependable and Secure Computing*, 15(4):591–606, 2018. doi:10.1109/TDSC.2016.2599873.
- 31 Yue Wang, Xintao Wu, and Leting Wu. Differential Privacy Preserving Spectral Graph Analysis. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 329–340, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-37456-2_28.

- 32 Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Private Release of Graph Statistics using Ladder Functions. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 731–745, New York, NY, USA, May 2015. Association for Computing Machinery. doi:10.1145/2723372.2737785.
- 33 Sen Zhang, Weiwei Ni, and Nan Fu. Differentially Private Graph Publishing with Degree Distribution Preservation. *Computers & Security*, page 102285, 2021. doi:10.1016/j.cose.2021.102285.

Experimental Comparison of PC-Trees and PQ-Trees

Simon D. Fink 

Faculty of Informatics and Mathematics, Universität Passau, Germany

Matthias Pfretzschner 

Faculty of Informatics and Mathematics, Universität Passau, Germany

Ignaz Rutter 

Faculty of Informatics and Mathematics, Universität Passau, Germany

Abstract

PQ-trees and PC-trees are data structures that represent sets of linear and circular orders, respectively, subject to constraints that specific subsets of elements have to be consecutive. While equivalent to each other, PC-trees are conceptually much simpler than PQ-trees; updating a PC-tree so that a set of elements becomes consecutive requires only a single operation, whereas PQ-trees use an update procedure that is described in terms of nine transformation templates that have to be recursively matched and applied.

Despite these theoretical advantages, to date no practical PC-tree implementation is available. This might be due to the original description by Hsu and McConnell [14] in some places only sketching the details of the implementation. In this paper, we describe two alternative implementations of PC-trees. For the first one, we follow the approach by Hsu and McConnell, filling in the necessary details and also proposing improvements on the original algorithm. For the second one, we use a different technique for efficiently representing the tree using a Union-Find data structure. In an extensive experimental evaluation we compare our implementations to a variety of other implementations of PQ-trees that are available on the web as part of academic and other software libraries. Our results show that both PC-tree implementations beat their closest fully correct competitor, the PQ-tree implementation from the OGDF library [6, 15], by a factor of 2 to 4, showing that PC-trees are not only conceptually simpler but also fast in practice. Moreover, we find the Union-Find-based implementation, while having a slightly worse asymptotic runtime, to be twice as fast as the one based on the description by Hsu and McConnell.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Mathematics of computing → Trees; Mathematics of computing → Graph algorithms

Keywords and phrases PQ-Tree, PC-Tree, circular consecutive ones, implementation, experimental evaluation

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.43

Related Version *Full Version:* <https://arxiv.org/abs/2106.14805>

Supplementary Material *Software (Source Code):* <https://github.com/N-Coder/pc-tree/>
archived at `swb:1:dir:00616576c1a16938cc5c376685086c7c2c368f7d`

Funding Work partially supported by DFG-grant Ru-1903/3-1.

Simon D. Fink: DFG-grant Ru-1903/3-1.

Matthias Pfretzschner: DFG-grant Ru-1903/3-1.

Ignaz Rutter: DFG-grant Ru-1903/3-1.

1 Introduction

PQ-trees represent linear orders of a ground set subject to constraints that require specific subsets of elements to be consecutive. Similarly, PC-trees do the same for circular orders subject to consecutivity constraints. PQ-trees were developed by Booth and Lueker [3] to



© Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 43; pp. 43:1–43:13

Leibniz International Proceedings in Informatics



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

solve the consecutive ones problem, which asks whether the columns of a Boolean matrix can be permuted such that the 1s in each row are consecutive. PC-trees are a more recent generalization introduced by Shih and Hsu [16] to solve the circular consecutive ones problem, where the 1s in each row only have to be circularly consecutive.

Though PQ-trees represent linear orders and PC-trees represent circular orders, Haeupler and Tarjan [10] show that in fact PC-trees and PQ-trees are equivalent, i.e., one can use one of them to implement the other without affecting the asymptotic running time. The main difference between PQ-trees and PC-trees lies in the update procedure. The update procedure takes as input a PQ-tree (a PC-tree) T and a subset U of its leaves and produces a new PQ-tree (PC-tree) T' that represents exactly the linear orders (circular orders) represented by T where the leaves in U appear consecutively. The update procedure for PC-trees consists only of a single operation that is applied independently of the structure of the tree. In contrast, the update of the PQ-tree is described in terms of a set of nine template transformations that have to be recursively matched and applied.

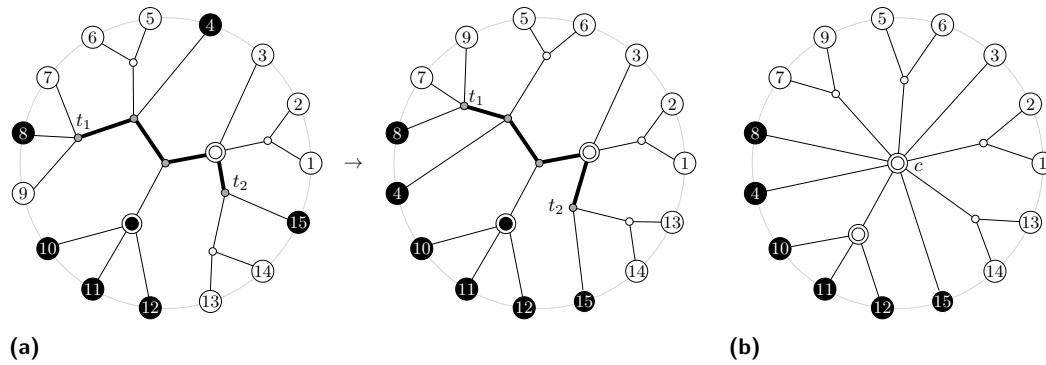
PQ-trees have numerous applications, e.g., in planarity testing [3, 16], recognition of interval graphs [3] and genome sequencing [1]. Nevertheless, PC-trees have been adopted more widely, e.g., for constrained planarity testing problems [2, 5] due to their simpler update procedure. Despite their wide applications and frequent use in theoretical algorithms, few PQ-tree implementations and even fewer PC-tree implementations are available. Table 1 shows an overview of all PC/PQ-tree implementations that we are aware of, though not all of them are working.

In this paper we describe the first correct and generic implementations of PC-trees. Section 2 contains an overview of the update procedure for applying a new restriction to a PC-tree. In Section 3, we describe the main challenge when implementing PC-trees and how our two implementations take different approaches at solving it. In Section 4, we present an extensive experimental evaluation, where we compare the performance of our implementations with the implementations of PC-trees and PQ-trees from Table 1. Our experiments show that, concerning running time, PC-trees following Hsu and McConnell's original approach beat their closest competitor, the PQ-tree implementation from the OGDF library [6] by roughly a factor 2. Our second implementation using Union-Find is another 50% faster than this first one, thus beating the OGDF implementation by a factor of up to 4.

2 The PC-tree

A *PC-tree* T is a tree without degree-2 vertices whose inner nodes are partitioned into *P-nodes* and *C-nodes*. Edges incident to C-nodes have a circular order that is fixed up to reversal, whereas edges incident to P-nodes can be reordered arbitrarily. Traversing the tree according to fixed orders around the inner nodes determines a circular ordering of the leaves L of the tree. Any circular permutation of L that can be obtained from T after arbitrarily reordering the edges around P-nodes and reversing orders around C-nodes is a *valid permutation* of L . In this way a PC-tree represents a set of circular permutations of L .

When applying a *restriction* $R \subseteq L$ to T , we seek a new tree that represents exactly the valid permutations of L where the leaves in R appear consecutively. We call a restriction *impossible* if there is no valid permutation of L where the leaves in R are consecutive. Thus, restriction R is possible if and only if the edges incident to P-nodes can be rearranged and orders of edges incident to C-nodes can be reversed in such a way that all leaves in R are consecutive. Updating a PC-tree to enforce the new restriction can thus be done by identifying and adapting the nodes that decide about the consecutivity of the elements of R and then changing the tree to ensure that this consecutivity can no longer be broken.

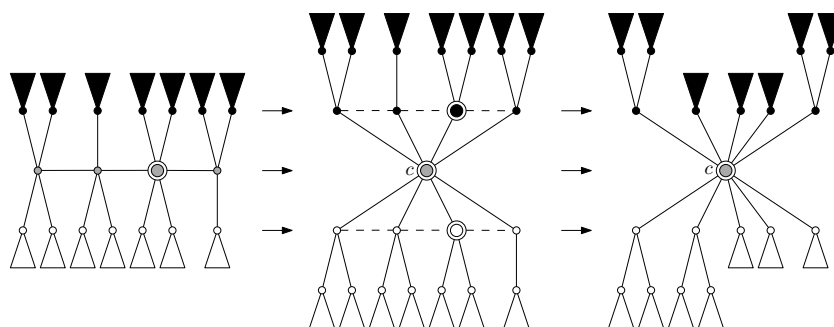


■ **Figure 1** (a) Two equivalent PC-Trees with their nodes colored according to the restriction $\{4, 8, 10, 11, 12, 15\}$. C-nodes are represented by big double circles and the P-nodes are represented by small circles. The white nodes represent empty nodes, the black nodes represent full nodes and the gray nodes represent partial nodes. The thick edges represent the terminal path with terminal nodes t_1 and t_2 . As the restriction is possible, all full leaves of the tree on the left can be made consecutive, as shown on the right. Furthermore all nodes that must be modified lie on a path. (b) Updated PC-tree with new central C-node c .

Let a leaf $x \in L$ be *full* if $x \in R$ and *empty* otherwise. We call an edge *terminal* if the two subtrees separated by the edge both contain at least one empty and at least one full leaf. Exactly the endpoints of all terminal edges need to be “synchronized” to ensure that all full leaves are consecutive. Hsu and McConnell [14, 13] show that R is possible if and only if the terminal edges form a path and all nodes of this path can be flipped so that all full leaves are on one side and all empty leaves are on the other. This path is called the *terminal path*, the two nodes at the ends of the terminal path are the *terminal nodes*. Observe that each node in T that is adjacent to two subtrees of which one only contains full leaves and the other contains only empty leaves is contained in the terminal path. Figure 1a illustrates the terminal path.

When updating T in order to apply the restriction, every node on the terminal path is split into two nodes, one of which holds all edges to neighbors of the original node whose subtree has only full leaves, the other holds all edges to empty neighbors, while terminal edges are deleted. A new central C-node c is created that is adjacent to all the split nodes in such a way that it preserves the order of the neighbors around the terminal path. Contracting all edges to the split C-nodes incident to c and contracting all nodes with degree two results in the updated tree that represents the new restriction [14, 13]. Figure 1 shows an example of this update, while Figure 2 details changes made to the terminal path.

It remains to efficiently find the terminal edges, and thus the subtrees with mixed full and empty leaves. To do so, Hsu and McConnell first choose an arbitrary node of the tree as root. They also assign labels to the inner nodes of the tree, marking an inner node (and conceptually the subtree below it) *partial* if at least one of its neighbors (i.e. children or parent) is full, *full* if all its neighbors except one (which usually is the parent) are full, and *empty* otherwise. Then, an edge is terminal if and only if it lies on a path between two partial nodes [14, 13]. Assigning the labels and subsequently finding the terminal edges can be done by two bottom-up traversals of the tree. We summarize these steps in the following, more fine-granular description of Hsu and McConnell’s algorithm for updating the PC-tree [13, Algorithm 32.2]:



■ **Figure 2** Left: The terminal path with all full subtrees shown in black on top and all empty subtrees shown in white on the bottom. Middle: The updated PC-tree, where all terminal edges were deleted, all nodes on the terminal were split in a full and empty half and all new nodes were connected to a new C-node c . Right: The PC-Tree after contracting all new C-nodes and all degree-2 P-nodes into c .

Algorithm for Applying Restrictions. To add a new restriction R to a PC-tree T :

1. Label all partial and full nodes by searching the tree bottom-up from all full leaves.
2. Find the terminal path by walking the tree upwards from all partial nodes in parallel.
3. Perform flips of C-nodes and modify the cyclic order of edges incident to P-nodes so that all full leaves lie on one side of the path.
4. Split each node on the path into two nodes, one incident to all edges to full leaves and one incident to all edges to empty leaves.
5. Delete the edges of the path and replace them with a new C-node c , adjacent to all split nodes, whose cyclic order preserves the order of the nodes on this path.
6. Contract all edges from c to adjacent C-nodes, and contract any node that has only two neighbors.

3 Our Implementations

The main challenge posed to the data structure for representing the PC-tree is that, in step 6, it needs to be able to merge arbitrarily large C-nodes in constant time for the overall algorithm to run in linear time. This means that, whenever C-nodes are merged, updating the pointer to a persistent C-node object on every incident edge would be too expensive. Hsu and McConnell (see [13, Definition 32.1]) solve this problem by using C-nodes that, instead of having a permanent node object, are only represented by the doubly-linked list of their incident half-edges, which we call *arcs*. This complicates various details of the implementation, like finding the parent pointer of a C-node, which are only superficially covered in the initial work of Hsu and McConnell [14]. These issues are in part remedied by the so called *block-spanning pointers* introduced in the later published book chapter [13], which are related to the pointer borrowing strategy introduced by Booth and Lueker [3]. These block-spanning pointers link the first and last arc of a consecutive block of full arcs (i.e. the arcs to full neighbors) around a C-node and can be accompanied by temporary C-node objects. Whenever a neighbor of a C-node becomes full, either a new block is created for the corresponding arc of the C-node, an adjacent block grows by one arc, or the two blocks that now became adjacent are merged.

Using this data structure, Hsu and McConnell show that the addition of a single new restriction R takes $O(p + |R|)$ time, where p is the length of the terminal path, and that applying restrictions R_1, \dots, R_k takes $\Theta(|L| + \sum_{i=1}^k |R_i|)$ time [14, 13]. Especially for steps 1 and 2, they only sketch the details of the implementation, making it hard to directly put

it into practice. In the full version, we fill in the necessary details for these steps and also refine their runtime analysis, showing that step 1 can be done in $O(|R|)$ time and step 2 can be done in $O(p)$ time. Using the original procedures by Hsu and McConnell, steps 3 and 4 can be done in $O(|R|)$ time and steps 5 and 6 can be done in $O(p)$ time.

For our first implementation, which we call HsuPC, we directly implemented these steps in C++, using the data structure without permanent C-node objects as described by Hsu and McConnell. During the evaluation, we realized that traversals of the tree are expensive. This is plausible, as they involve a lot of pointer-dereferencing to memory segments that are not necessarily close-by, leading to cache misses. To avoid additional traversals for clean-up purposes, we store information that is valid only during the update procedure with a timestamp. Furthermore, we found that keeping separate objects for arcs and nodes and the steps needed to work around the missing C-node objects pose a non-negligible overhead.

To remove this overhead, we created a second version of our implementation, which we call UFPC, using a Union-Find tree for representing C-node objects: Every C-node is represented by an entry in the Union-Find tree and every incident child edge stores a reference to this entry. Whenever two C-nodes are merged, we apply `union` to both entries and only keep the object of the entry that survives. This leads to every lookup of a parent C-node object taking amortized $O(\alpha(|L|))$ time, where α is the inverse Ackermann function. Although this makes the overall runtime super-linear, the experimental evaluation following in the next section shows that this actually improves the performance in practice. As a second change, the UFPC no longer requires separate arc and node objects, allowing us to use a doubly-linked tree consisting entirely of nodes that store pointers to their parent node, left and right sibling node, and first and last child node. Edges are represented implicitly by the child node whose parent is the other end of the edge. Note that of the five stored pointers, a lookup in the Union-Find data structure is only needed for resolving the parent of a node.

Our algorithmic improvements and differences of both implementations are described in more detail in the full version. We use the Union-Find data structure from the OGDF [6] and plan to merge our UFPC implementation into the OGDF. Furthermore, both implementations should also be usable stand-alone with a custom Union-Find implementation. The source code for both implementations, our evaluation harness and all test data are available on GitHub (see Table 1).

4 Evaluation

In this section, we experimentally evaluate our PC-tree implementations by comparing the running time for applying a restriction with that of various PQ- and PC-tree implementations that are publicly available. In the following we describe our method for generating test cases, our experimental setup and report our results.

4.1 Test Data Generation

To generate PQ-trees and restrictions on them, we make use of the planarity test by Booth and Lueker [3], one of the initial applications of PQ-trees. This test incrementally processes vertices one by one according to an *st*-ordering. Running the planarity test on a graph with n vertices applies $n - 1$ restrictions to PQ-trees of various sizes. Since not all implementations provide the additional modification operations necessary to implement the planarity test, we rather export, for each step of the planarity test, the current PQ-tree and the restriction that is applied to it as one instance of our test set. We note that the use of *st*-orderings ensures that the instances do not require the ability of the PC-tree to represent circular permutations, making them good test cases for comparing PC-trees and PQ-trees.

43:6 Experimental Comparison of PC-Trees and PQ-Trees

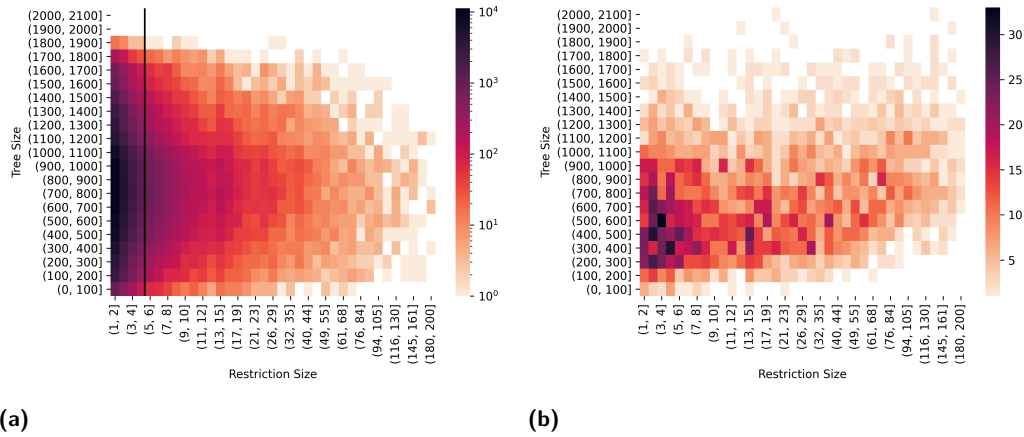


Figure 3 Distribution of tree and restriction size for the data sets **(a) SER-POS** and **(b) SER-IMP**. Please note the different color scales. The **SER-POS** instances that are left of the black line are too small and filtered out.

In this way, we create one test set **SER-POS** consisting of only PQ-trees with possible restrictions by exporting the instances from running the planarity test on a randomly generated biconnected planar graph for each vertex count n from 1000 to 20,000 in steps of 1000 and each edge count $m \in \{2n, 3n - 6\}$. To avoid clutter, to remedy the tendency of the planarity test to produce restrictions with very few leaves, and to avoid bias from trivial optimizations such as filtering trivial restrictions with $|R| \in \{1, |L| - 1, |L|\}$, which is present in some of the implementations, we filter test instances where $|R|$ lies outside the interval $[5, |L| - 2]$. Altogether, this test set contains 199,831 instances, whose distribution with regards to tree and restriction size is shown in Figure 3a.

To guard against overly permissive implementations, we also create a small test set **SER-IMP** of impossible restrictions. It is generated in the same way, by adding randomly chosen edges to the graphs from above until they become non-planar. In this case the planarity test fails with an impossible restriction at some point; we include these 3,800 impossible restrictions in the set, see Figure 3b.

As most of the available implementations have no simple means to store and load a PQ-/PC-tree, we serialize each test instance as a set of restrictions that create the tree, together with the additional new restriction. When running a test case, we then first apply all the restrictions to reobtain the tree, and then measure the time to apply the new restriction from the test case. The prefix **SER-** in the name of both sets emphasizes this serialization.

To be able to conduct a more detailed comparison of the most promising implementations, we also generate a third test set with much larger instances. As deserializing a PC- or PQ-tree is very time-consuming, we directly use the respective implementations in the planarity test by Booth and Lueker [3], thus calling the set **DIR-PLAN**. We generated 10 random planar graphs with n vertices and m edges for each n ranging from 100,000 to 1,000,000 in steps of 100,000 and each $m \in \{2n, 3n - 6\}$, yielding 200 graphs in total. The planarity test then yields one possible restriction per node. As we only want to test big restrictions, we filter out restrictions with less than 25 full leaves, resulting in **DIR-PLAN** containing 564,300 instances.

Table 1 Implementations considered for the evaluation. Implementations that are entirely unusable as they are incomplete or crash/produce incorrect results on almost all inputs (marked with $-$) and those where no stand-alone PC-/PQ-tree implementation could be extracted (marked with n.a.) could not be evaluated. Correct implementations are marked with \checkmark and implementations that are functional, but do not always produce correct results are marked with \times . These two categories are included in our experimental evaluation. The last column shows how many of 203,630 restrictions in the sets SER-POS and SER-IMP failed.

Name	Type	Context	Language	Correct	Errors	URL
HsuPC	PC-Tree	our impl., based on [13]	C++	\checkmark	0	https://github.com/N-Coder/pc-tree/tree/HsuPCSubmodule
UFPC	PC-Tree	our impl. using Union-Find	C++	\checkmark	0	https://github.com/N-Coder/pc-tree
Luk&Zhou	PC-Tree	student course project	C++	$-$	$-$	https://github.com/kwmichaelluk/pc-tree
Hsu [12]	PC-Tree	planarity test prototype	C++	n.a.	$-$	http://qa.iis.sinica.edu.tw/graphtheory
Noma [4]	PC-Tree	planarity test evaluation	C++	n.a.	$-$	https://www.ime.usp.br/~noma/sh
OGDF [15]	PQ-Tree	planarity testing	C++	\checkmark	0	https://ogdf.github.io
Gregable	PQ-Tree	biclustering	C++	\checkmark	0	https://gregable.com/2008/11/pq-tree-algorithm.html
BiVoC [9]	PQ-Tree	automatic layout of biclusters	C++	\times	71	https://bioinformatics.cs.vt.edu/~murali/papers/BiVoC
Reisle	PQ-Tree	student project	C++	\times	236	https://github.com/creisle/pq-trees
GraphSet [8]	PQ-Tree	visual graph editor	C++	\times	580	http://graphset.cs.arizona.edu
Zanetti [17]	PQR-Tree ¹	extension of PQ-Trees	Java	\times	454	https://github.com/jppzanetti/PQRTree
CppZanetti	PQR-Tree ¹	our C++ conversion of Zanetti	C++	\times	454	https://github.com/N-Coder/pc-tree#installation
JGraphEd [11]	PQ-Tree	visual graph editor	Java	\times	11	https://www3.cs.stonybrook.edu/~algorith/implementation/jgraphed/implementation.shtml
GTea [7]	PQ-Tree	visual graph theory tool	Java	$-$	$-$	https://github.com/rostaam/GTea
TryAlgo	PQ-Tree	consecutive-ones testing	Python	$-$	$-$	https://tryalgo.org/en/datastructures/2017/12/15/pq-trees
SageMath	PQ-Tree	interval graph detection	Python	\checkmark	0	https://doc.sagemath.org/html/en/reference/graphs/sage/graphs/pq_trees.html

¹ PQR-Trees are a variant of PQ-Trees that can also represent impossible restrictions, replacing any node that would make a restriction impossible by an R-node (again allowing arbitrary permutation). To make the implementations comparable, we abort early whenever an impossible restriction is detected and an R-node would be generated.

4.2 Experimental Setup

Table 1 gives an overview of all implementations we are aware of, although not all implementations could be considered for the evaluation.

The three existing implementations of PC-trees we found are incomplete and unusable (Luk&Zhou) or tightly intertwined with a planarity test in such a way that we were not able to extract a generic implementation of PC-trees (Hsu, Noma). We further exclude two PQ-tree implementations as they either crash or produce incorrect results on almost all inputs (GTea) or have an excessively poor running time (TryAlgo). Among the remaining PQ-tree implementations only three correctly handle all our test cases (OGDF, Gregable, SageMath). Several other implementations have smaller correctness issues: After applying a fix to prevent segmentation faults in a large number of cases for BiVoC, the remaining implementations crash (BiVoC, GraphSet, Zanetti) and/or produce incorrect results (Reisle, JGraphEd, Zanetti) on a small fraction of our tests; compare the last column of Table 1. We nevertheless include them in our evaluation.

We changed the data structure responsible for mapping the input to the leaves of the tree for BiVoC and Gregable from `std::map` to `std::vector` to make them competitive. Moreover, BiVoC, Gregable and GraphSet use a rather expensive cleanup step that has to be executed after each update operation. As this could probably largely be avoided by the use of timestamps, we do not include the cleanup time in their reported running times. For SageMath the initial implementation turned out to be quadratic, which we improved to linear by removing unnecessary recursion. As Zanetti turned out to be a close competitor to our implementation in terms of running time, we converted the original Java implementation to C++ to allow a fair comparison. This decreased the runtime by one third while still producing the exact same results. All other non-C++ implementations were much slower or had other issues, making a direct comparison of their running times within the same language environment as our implementations unnecessary. Further details on the implementations can be found in the full version.

Each experiment was run on a single core of a Intel Xeon E5-2690v2 CPU (3.00 GHz, 10 Cores) with 64 GiB of RAM, running Linux Kernel version 4.19. Implementations in C++ were compiled with GCC 8.3.0 and optimization `-O3 -march=native -mtune=native`. Java implementations were executed on OpenJDK 64-Bit Server VM 11.0.9.1 and Python implementations were run with CPython 3.7.3. For the Java implementations we ran each experiment several times, only measuring the last one to remove startup-effects and to facilitate optimization by the JIT compiler. We used OGDF version 2020.02 (Catalpa) to generate the test graphs.

4.3 Results

Our experiments turn out that SageMath, even with the improvements mentioned above, is on average 30 to 100 times slower than all other implementations.² For the sake of readability, we scale our plots to focus on the other implementations. As the main application of PC-/PQ-trees is applying possible restrictions, we first evaluate on the dataset SER-POS. Figure 4 shows the runtime for individual restrictions based on the size of the restriction (i.e. the number of full leaves) and the overall size of the tree. Figure 4a clearly shows that for all implementations the runtime is linear in the size of the restriction. Figure 4b suggests that

² Part of this might be due to the overhead of running the code with CPython. As the following analysis shows, SageMath also has other issues, allowing us to safely exclude it.

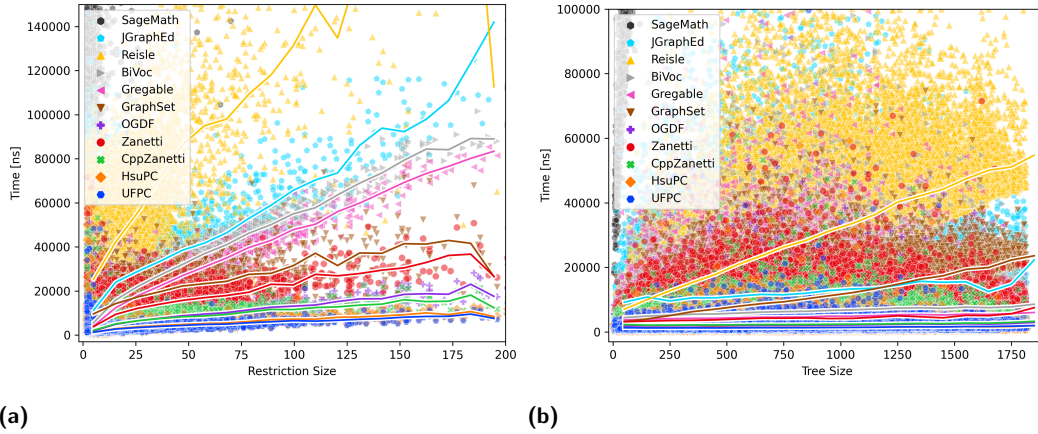


Figure 4 Runtime for SER-POS restrictions depending on (a) restriction size and (b) tree size.

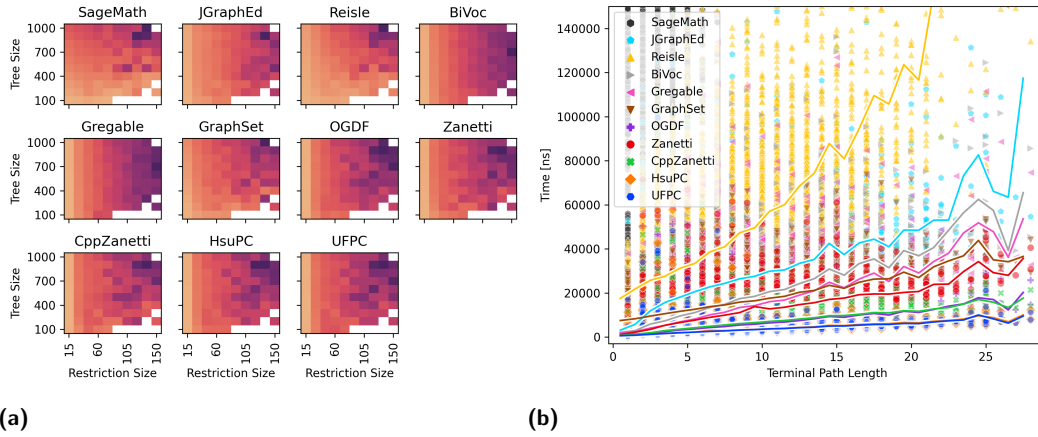


Figure 5 (a) A heatmap showing the average runtime of SER-POS restrictions, depending on both the size of the restriction and the size of the tree. The color scale is based on the maximum runtime of each respective implementation. (b) Runtime for SER-POS restrictions depending on the terminal path length.

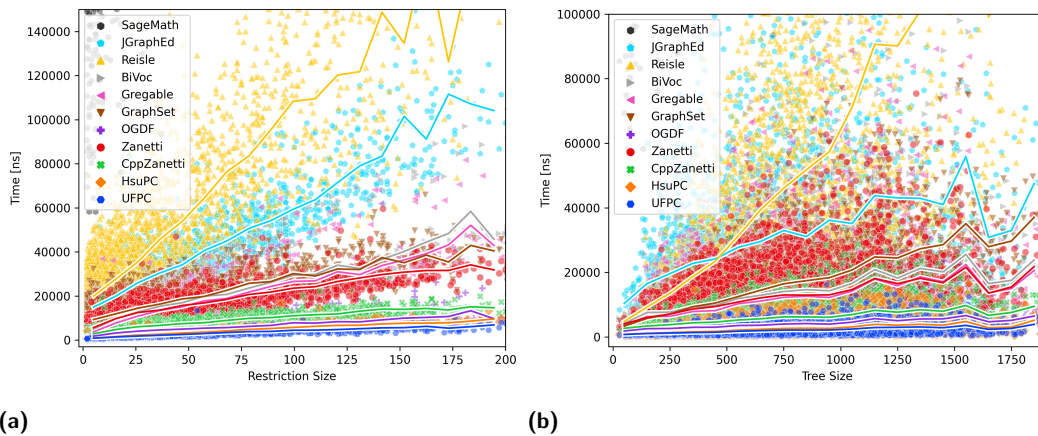


Figure 6 Runtime for SER-IMP restrictions depending on (a) restriction size and (b) tree size for all implementations.

the runtime of Reisle and GraphSet does not solely depend on the restriction size, but also on the size of the tree. To verify this, we created for each implementation a heatmap that indicates the average runtime depending on both the tree size and the restriction size, shown in Figure 5a. The diagonal pattern shown by SageMath, Reisle, and GraphSet confirms the dependency on the tree size. All other implementations exhibit vertical stripes, which shows that their runtime does not depend on the tree size. Finally, Figure 5b shows the runtime compared to the terminal path length. As expected, all implementations show a linear dependency on the terminal path length, with comparable results to Figure 4a.

Figure 6 shows the performance on the dataset **SER-IMP**. The performance is comparable with that on **SER-POS**. Noteworthy is that Zanetti performs quite a bit worse, which is due to its implementation not being able to detect failure during a labeling step. It always performs updates until a so-called R-node would be generated. Altogether, the data from **SER-POS** and **SER-IMP** shows that the implementations GraphSet, OGDF, Zanetti, HsuPC and UFPC are clearly superior to the others. In the following, we conduct a more detailed comparison of these implementations by integrating them into a planarity test and running them on much larger instances, i.e., the data set **DIR-PLAN**. In addition to an update method, this requires a method for replacing the now-consecutive leaves by a P-node with a given number of child leaves. Adding the necessary functionality would be a major effort for most of the implementations, which is why we only adapted the most efficient implementations to run this set. We also exclude GraphSet from this experiment; the fact that it scales linearly with the tree size causes the planarity test to run in quadratic time. Figure 7 again shows the runtime of individual restrictions depending on the restriction size. Curiously, Zanetti produces incorrect results for nearly all graphs with $m = 2n$ in Figure 7a. As the initial tests already showed, the implementation has multiple flaws; one major issue is already described in an issue on GitHub and another independent error is described in the full version. Both plots show that HsuPC is more than twice as fast as OGDF and that UFPC is again close to two times faster than HsuPC. Zanetti's runtime is roughly the same as that of HsuPC, while converting its Java code to C++ brings the runtime down close to that of UFPC.

As OGDF is the slowest, we use it as baseline to calculate the speedup of the other implementations. Figure 8a shows that the runtime improvement for all three implementations is the smallest for small restrictions, quickly increasing to the final values of roughly 0.4 times the runtime of OGDF for HsuPC and 0.25 for both CppZanetti and UFPC. Figure 8b shows the speedup depending on the length of the terminal path. For very short terminal paths (which are common in our datasets), both implementations are again close; but already for slightly longer terminal paths UFPC quickly speeds up to being roughly 20% faster than CppZanetti. This might be because creating the central node in step 5 is more complicated for UFPC, as the data structure without edge objects does not allow arbitrarily adding and removing edges (which is easier for HsuPC) and allowing circular restrictions forces UFPC to also pay attention to various special cases (which are not necessary for PQ-trees).

5 Conclusion

In this paper we have presented the first fully generic and correct implementations of PC-trees. One implementation follows the original description of Hsu and McConnell [14, 13], which contains several subtle mistakes in the description of the labeling and the computation of the terminal path. This may be the reason why no fully generic implementation has been available so far. A corrected version that also includes several small simplifications is described in the full version of this paper.

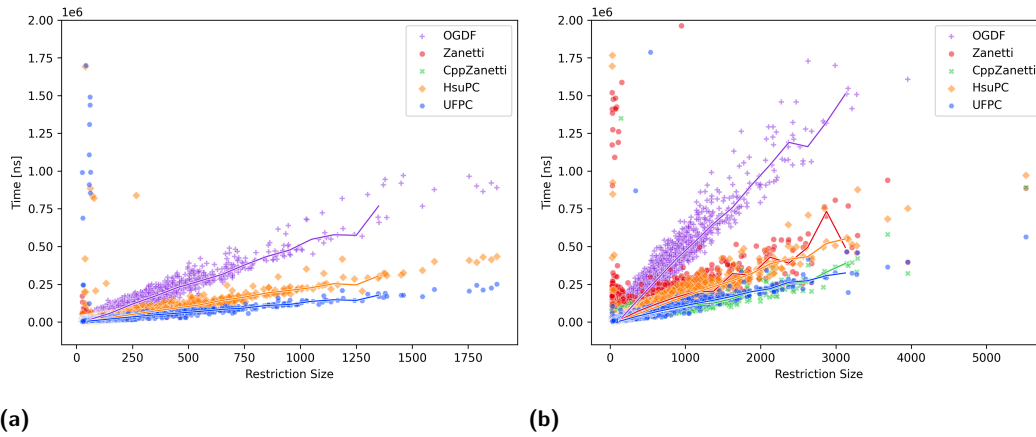


Figure 7 Runtime of individual restrictions of DIR-PLAN with OGDF, Zanetti and our implementations for graphs of size (a) $m = 2n$ and (b) $m = 3n - 6$.

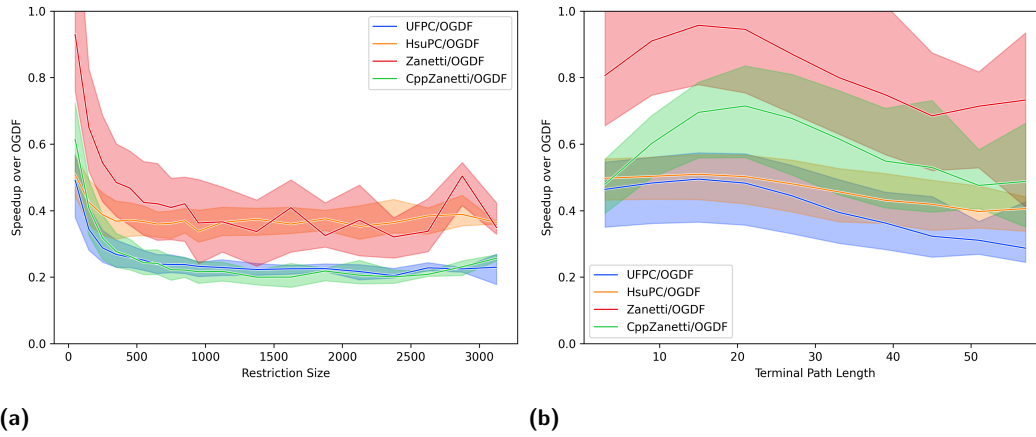


Figure 8 Median performance increase depending on (a) the size of the restriction and (b) the terminal path length, with OGDF as baseline. The shaded areas show the interquartile range.

Furthermore, we provided a second, alternative implementation, using Union-Find to replace many of the complications of Hsu and McConnell's original approach. Technically, this increases the runtime to $O((|R| + p) \cdot \alpha(|L|))$, where α is the inverse Ackerman function. In contrast, our evaluations show that the Union-Find-based approach is even faster in practice, despite the worse asymptotic runtime.

Our experimental evaluation with a variety of other implementations reveals that surprisingly few of them seem to be fully correct. Only three other implementation have correctly handled all our test cases. The fastest of them is the PQ-tree implementation of OGDF, which our Union-Find-based PC-tree implementation beats by roughly a factor of 4. Interestingly, the Java implementation of PQR-trees by Zanetti achieves a similar speedup once ported to C++. However, Zanetti's Java implementation is far from correct and it is hard to say whether it is possible to fix it without compromising its performance.

Altogether, our results show that PC-trees are not only conceptually simpler than PQ-trees but also perform well in practice, especially when combined with Union-Find. To put the speedup of factor 4 into context, we compared the OGDF implementations of the

planarity test by Booth and Lueker and the one by Boyer and Myrvold on our graph instances. The Boyer and Myrvold implementation was roughly 40% faster than the one based on Booth and Lueker's algorithm. Replacing the PQ-trees, which are the core part of the latter, by an implementation that is 4 time faster, might make this planarity test run faster than the one by Boyer and Myrvold. We leave a detailed evaluation, also taking into account the embedding generation, which our PC-tree based planarity test not yet provides, for future work.

References

- 1 S. Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Sciences*, 45(11):1607–1620, November 1959. doi:10.1073/pnas.45.11.1607.
- 2 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Trans. Algorithms*, 12(2):16:1–16:46, 2016. doi:10.1145/2738054.
- 3 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.
- 4 John M. Boyer, Cristina G. Fernandes, Alexandre Noma, and José C. de Pina. Lempel, Even, and Cederbaum planarity method. In *Experimental and Efficient Algorithms*, pages 129–144. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24838-5_10.
- 5 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. In Philip N. Klein, editor, *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.
- 6 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 17. CRC Press, 2014.
- 7 Alex William Cregten and Hannes Kristján Hannesson. Implementation of a planarity testing method using PQ-trees. Technical report, Reykjavík University, 2017. URL: https://skemman.is/bitstream/1946/29618/1/Planarity_testing_with_PQTrees.pdf.
- 8 Alejandro Estrella-Balderrama, J. Joseph Fowler, and Stephen G. Kobourov. Graph simultaneous embedding tool, GraphSET. In *Graph Drawing*, pages 169–180. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00219-9_17.
- 9 Gregory A. Grothaus, Adeel Mufti, and T. M. Murali. Automatic layout and visualization of biclusters. *Algorithms for Molecular Biology*, 1(1):15, 2006. doi:10.1186/1748-7188-1-15.
- 10 Bernhard Haeupler and Robert E. Tarjan. Planarity algorithms via PQ-trees (extended abstract). *Electronic Notes in Discrete Mathematics*, 31:143–149, August 2008. doi:10.1016/j.endm.2008.06.029.
- 11 Jon Harris. JGraphEd – a java graph editor and graph drawing framework. Technical report, Carleton University, School of Computer Science, Comp 5901 Directed Studies, 2004. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.188.5066&rank=1>.
- 12 Wen-Lian Hsu. An efficient implementation of the PC-tree algorithm of Shih & Hsu's planarity test. Technical report, Institute of Information Science, Academia Sinica, 2003. URL: http://ias1.iis.sinica.edu.tw/webpdf/paper-2003-PLANAR_implementation.pdf.
- 13 Wen-Lian Hsu and Ross McConnell. PQ trees, PC trees, and planar graphs. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 32. Chapman and Hall/CRC, January 2004. doi:10.1201/9781420035179.
- 14 Wen-Lian Hsu and Ross M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, March 2003. doi:10.1016/s0304-3975(02)00435-8.
- 15 Sebastian Leipert. PQ-trees, an implementation as template class in C++. Technical report, University of Cologne, 1997. URL: <http://e-archive.informatik.uni-koeln.de/id/eprint/259>.

- 16 Wei-Kuan Shih and Wen-Lian Hsu. A new planarity test. *Theoretical Computer Science*, 223(1-2):179–191, July 1999. doi:10.1016/s0304-3975(98)00120-0.
- 17 João Paulo Pereira Zanetti. Complexidade de construção de árvores PQR. Master's thesis, Universidade Estadual de Campinas, Instituto de Computação, 2012. URL: http://bdtd.ibict.br/vufind/Record/CAMP_0b551865d78ef032289f17f95e3ccee7.

Boundary-Sensitive Approach for Approximate Nearest-Neighbor Classification

Alejandro Flores-Velazco ✉ 

Department of Computer Science, University of Maryland, College Park, MD, USA

David M. Mount ✉ 

Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD, USA

Abstract

The problem of *nearest-neighbor classification* is a fundamental technique in machine-learning. Given a training set P of n labeled points in \mathbb{R}^d , and an approximation parameter $0 < \varepsilon \leq \frac{1}{2}$, any unlabeled query point should be classified with the class of any of its ε -approximate nearest-neighbors in P . Answering these queries efficiently has been the focus of extensive research, proposing techniques that are mainly tailored towards resolving the more general problem of ε -approximate nearest-neighbor search. While the latest can only hope to provide query time and space complexities dependent on n , the problem of nearest-neighbor classification accepts other parameters more suitable to its analysis. Such is the number k_ε of ε -border points, which describes the complexity of boundaries between sets of points of different classes.

This paper presents a new data structure called Chromatic AVD. This is the first approach for ε -approximate nearest-neighbor classification whose space and query time complexities are only dependent on ε , k_ε and d , while being independent on both n and Δ , the spread of P .

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases approximate nearest-neighbor searching, nearest-neighbor classification, geometric data structures, space-time tradeoffs

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.44

Funding Research partially supported by NSF grant CCF-1618866.

1 Introduction

Non-parametric classification is a fundamental technique in machine-learning. In this context, we are given a *training set* P consisting of n points in a metric space $(\mathcal{X}, \mathbf{d})$, with domain \mathcal{X} and distance function $\mathbf{d} : \mathcal{X}^2 \rightarrow \mathbb{R}^+$. Additionally, the training set is partitioned into a finite set of *classes* C by associating each point $p \in P$ with a *label* $l(p)$, which indicates the class to which it belongs. Given an *unlabeled* query point $q \in \mathcal{X}$, the goal of a *classifier* is to predict q 's label using the training set P .

The *nearest-neighbor rule* is among the best-known classification techniques [15]. It assigns a query point the label of its closest point in P according to the defined metric. This technique exhibits good classification accuracy both experimentally and theoretically [12, 13, 34], but it is often criticized due to its high space and time complexities. Despite the advent of more sophisticated techniques (*e.g.*, support-vector machines [11] and deep neural networks [32]), nearest-neighbor classification is still widely used in practice [9, 22, 26], proving its value in constructing resilient defense strategies against adversarial [27] and poisoning [29] attacks, as well as in achieving interpretable machine-learning models [28, 33].

As mentioned, the criticism towards nearest-neighbor classification lingers on the bases of exceedingly high query times and space requirements. The standard approach to answering these queries, even approximately, involves storing the entire training set P or at least a



© Alejandro Flores-Velazco and David M. Mount;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 44; pp. 44:1–44:15

Leibniz International Proceedings in Informatics



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

sufficiently large part of it. This implies that the time spent to answer such queries depends to a large degree on the size and dimensionality of the training set of points stored, limiting the use of nearest-neighbor classification on large-scale applications.

In this paper we explore efficient methods for approximate nearest-neighbor classification. We are given the training set P and an approximation parameter $0 < \varepsilon \leq \frac{1}{2}$. The objective is to construct a data structure so that given any query point q , it is possible to efficiently classify q according to any valid ε -approximate nearest-neighbor in P . Throughout, we take the domain \mathcal{X} to be d -dimensional Euclidean space \mathbb{R}^d , the distance function d to be the L_2 norm, and we assume that the dimension d is a fixed constant, independent of n and ε .

1.1 Related Work

In the standard ε -approximate nearest-neighbor searching (ε -ANN), the objective is to compute a point whose distance from the query point is within a factor of $1 + \varepsilon$ of the true nearest neighbor. This problem, referred to as “standard ANN” throughout, has been widely studied. In *chromatic ε -ANN search* the objective is to return just the class (or more visually, the “color”) of any such point [6, 18]. We refer to this as ε -classification.

Clearly, chromatic ANN queries can be reduced to standard ANN queries. Hence, most of the efficiency improvements in nearest-neighbor classification have arisen from improvements to the standard ANN problem. While standard ANN has been well studied in high-dimensional spaces (see, e.g., [1]), in constant-dimensional Euclidean space, the most efficient data structures involve variants of the *Approximate Voronoi Diagram* (or AVD) (see [3–5, 23]). Arya *et al.* [6] proposed a data structure specifically tailored for ε -classification. Unfortunately, this work was based on older technology, and its results are not competitive when compared to the most recent advances on standard ANN search via AVDs.

All previous results have query and space complexities that depend on n , the total size of the training set P . In many cases, a much smaller portion of the training set may suffice to correctly ε -classify queries. Think of the boundaries between adjacent Voronoi cells of points of different classes (see Figure 1a). The points that define these boundaries are known as *border points*. Throughout, let k denote the number of such border points in P (clearly, $k \leq n$, and hopefully, $k \ll n$). Furthermore, the notion of border points can be generalized to the context of ε -classification (see Section 2 for a formal definition). Thus, denote k_ε as the number of ε -border points, where $k \leq k_\varepsilon \leq n$. Ideally, we would like the query and space complexities of answering chromatic ε -ANN queries to depend on k_ε instead of n .

In order to achieve this goal, previous research has focused on reducing the training set P by selecting a subset $R \subseteq P$. Once R is computed, it is assumed that this subset will be used to build a standard AVD for ε -classification. Research in this area is vast, but there are two broad approaches, depending on the type and size of the computed subsets, and the classification guarantees provided.

Heuristics: Most of the work has focused on proposing heuristics to compute smaller training sets $R \subseteq P$. These are often known as *condensation algorithms*, and the literature on these is extensive (see [25, 35] for comprehensive surveys, and [2, 7, 8, 21, 24, 30] for some of the proposed algorithms). The most recent condensation algorithms [16, 17, 20] show that it is possible to compute subsets of P of size $\mathcal{O}(k)$ in $\mathcal{O}(n^2)$ time. However, when AVDs are built from these subsets, the resulting data structures are likely to introduce classification errors [19], especially for query points that should be easily ε -classified. Thus, while often used in practice, these approaches do not guarantee that chromatic ε -ANN queries are answered correctly.

Coresets: Recent results propose a technique to compute a coreset for ε -classification [19].

A coreset R guarantees that every query point will be correctly classified when assigning the class of the point of R returned by the AVD. That is, for any query $q \in \mathbb{R}^d$, the point of R returned by the AVD belongs to the same class as one of q 's ε -approximate nearest-neighbors in P . Unfortunately, the size of the computed coreset can be as large as $\mathcal{O}((k \log \Delta)/\varepsilon^{d-1})$, where Δ is the spread¹ of P .

1.2 Contributions

From the previous section, we have seen that existing approaches for ε -classification achieve only one of the following goals:

- The size of the resulting data structure is dependent only on ε , k_ε (the number of ε -border points) and d , while being independent from n and Δ .
- It guarantees correct ε -classification for any query point.

The main result of this paper is an approach that achieves both goals. We propose a new data structure built specifically to answer chromatic ε -ANN queries over the training set P , which we call a *Chromatic AVD*. Given any query point $q \in \mathbb{R}^d$, this data structure returns the class to be assigned to q , which matches the class of at least one of q 's ε -approximate nearest-neighbors in P . More generally, our data structure returns a set of classes such that there is an ε -approximate nearest-neighbor of q from each of these classes.

Therefore, the Chromatic AVD can be used to correctly ε -classify any query point. The main result of this work is summarized in the following theorem, expressed in the form of a space-time tradeoff based on a parameter γ .

► **Theorem 1.** *Given a training set P of n labeled points in \mathbb{R}^d , an error parameter $0 < \varepsilon \leq \frac{1}{2}$, and a separation parameter $2 \leq \gamma \leq \frac{1}{\varepsilon}$. Let k_ε be the number of ε -border points of P . There exists a data structure for ε -classification, called *Chromatic AVD*, with:*

$$\text{Query time: } \mathcal{O}\left(\log(k_\varepsilon \gamma) + \frac{1}{(\varepsilon \gamma)^{\frac{d-1}{2}}}\right) \quad \text{Space: } \mathcal{O}\left(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon}\right).$$

Which can be constructed in time $\tilde{\mathcal{O}}\left(\left(n + k_\varepsilon / (\varepsilon \gamma)^{\frac{3}{2}(d-1)}\right) \gamma^d \log \frac{1}{\varepsilon}\right)$.

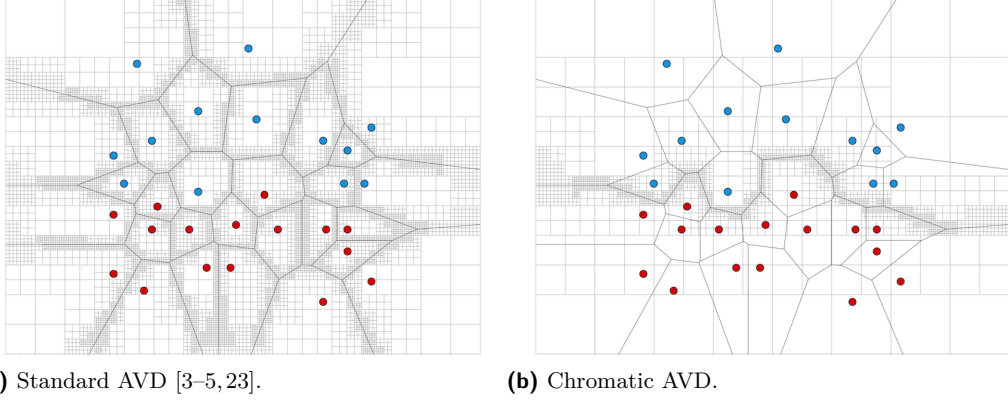
By setting γ to either of its extreme values, we obtain the following query times and space complexities.

► **Corollary 2.** *The separation parameter γ describes the tradeoffs between the query time and space complexity of the Chromatic AVD. This yields the following results:*

$$\begin{aligned} \text{If } \gamma = 2 & \longrightarrow \text{Query time: } \mathcal{O}\left(\log k_\varepsilon + \frac{1}{\varepsilon^{\frac{d-1}{2}}}\right) & \text{Space: } \mathcal{O}\left(k_\varepsilon \log \frac{1}{\varepsilon}\right). \\ \text{If } \gamma = \frac{1}{\varepsilon} & \longrightarrow \text{Query time: } \mathcal{O}\left(\log \frac{k_\varepsilon}{\varepsilon}\right) & \text{Space: } \mathcal{O}\left(\frac{k_\varepsilon}{\varepsilon^d}\right). \end{aligned}$$

The approach towards constructing this data structure is hybrid, combining a quadtree-induced partitioning of space (leveraging similar techniques to the ones used for standard AVDs), with the construction of coresets for only some cells of this partition. All other cells can be discarded, and a new quadtree can be built with only the remaining cells. The final size of the tree is bounded in terms of k_ε . This technique allows us to maintain coresets in the most critical regions of space, and thus, avoiding the dependency on the spread of P .

¹ The *spread* of a point set is defined to be the ratio between the largest and smallest pairwise distances.



■ **Figure 1** Examples of the space partitioning achieved by any standard AVD, compared to the Chromatic AVD data structure proposed in this paper. Our approach subdivides the space around the boundaries defined by the ε -border points, while ignoring other boundaries.

2 Preliminary Ideas and Intuition

Preliminaries. First, we need to introduce some preliminary definitions and notations that are relevant to the results presented in the remaining of the paper. Given any point $q \in \mathbb{R}^d$, denote its nearest-neighbor as $\text{nn}(q)$, and the distance between them by $d_{\text{nn}}(q) = d(q, \text{nn}(q))$.

Additionally, let's introduce a few concepts and related properties that will prove useful in the construction of the Chromatic AVD. These are Well-Separated Pair Decompositions [10] (WSPDs), Quadtrees [14, 31], and Approximate Voronoi Diagrams [3-5, 23] (AVDs).

Well-Separated Pair Decompositions: Given the point set P , and a separation factor $\sigma > 2$, we say that two sets $X, Y \subseteq P$ are *well separated* if they can be enclosed within two disjoint balls of radius r , such that the distance between the centers of these balls is at least σr . We say that X and Y form a *dumbbell*, where both sets are the *heads* of this dumbbell. Consider the line segment that connects the centers of both balls, and let z and ℓ be the center and length of this line segment, respectively (*i.e.*, the center and the length of the dumbbell). The following properties hold when $\sigma > 4$, for $x \in X$ and $y \in Y$:

$$d(x, z) < \ell \quad \ell < 2d(x, y) \quad \ell > d(x, y)/2.$$

Furthermore, a well-separated pair decomposition of P is defined as a set $\mathcal{D} = \{(X_i, Y_i)\}_i$ where every X_i and Y_i are well separated, and for every two distinct points $p_1, p_2 \in P$ there exists a unique pair $\mathcal{P} = (X, Y) \in \mathcal{D}$ such that $p_1 \in X$ and $p_2 \in Y$, or vice-versa. It is known how to construct a WSPD of P with $\mathcal{O}(\sigma^d n)$ pairs in $\mathcal{O}(n \log n + \sigma^d n)$ time.

Quadrees: These are tree data structures that provide a hierarchical partition of space. Each node in this tree consists of a d -dimensional hypercube, where non-leaf nodes partition its corresponding hypercube into 2^d equal parts. The root of this tree corresponds to the $[0, 1]^d$ hypercube. We will use a variant of this structure called a *balanced box-decomposition tree* (BBD tree) [6]. Such data structure satisfies the following properties:

1. Given a point set P , such a tree can be built in $\mathcal{O}(n \log n)$ time, having space $\mathcal{O}(n)$ such that each leaf node contains at most one point of P .
2. Given a collection \mathcal{U} of n quadtree boxes in $[0, 1]^d$, such a tree can be built in $\mathcal{O}(n \log n)$ time, having $\mathcal{O}(n)$ nodes such that the subdivision induced by its leaf cells is a refinement of the subdivision induced by the Quadtree boxes in \mathcal{U} .
3. Given the trees from 1 or 2, it is possible to determine the leaf cell containing any arbitrary query point q in $\mathcal{O}(\log n)$ time.

Approximate Voronoi Diagram: Generally, AVDs are quadtree-based data structures that can be used to efficiently answer ANN queries. The partitioning of space induced by this data structure is often generated from a WSPD of P . Additionally, every leaf cell w of this quadtree has an associated set of ε -representatives R_w that has the following property: for any query point $q \in w$, at least one point in R_w is one of q 's ε -approximate nearest-neighbors in P .

New Ideas and Intuitions. Consider the space partitioning induced by a standard AVD, as previously described. By construction, any leaf cell w of this partition has an associated set of ε -representatives R_w . Evidently, for the purposes of ε -classification, the most important information related to this leaf cell comes from the classes of the points in R_w , and not necessarily the points themselves.

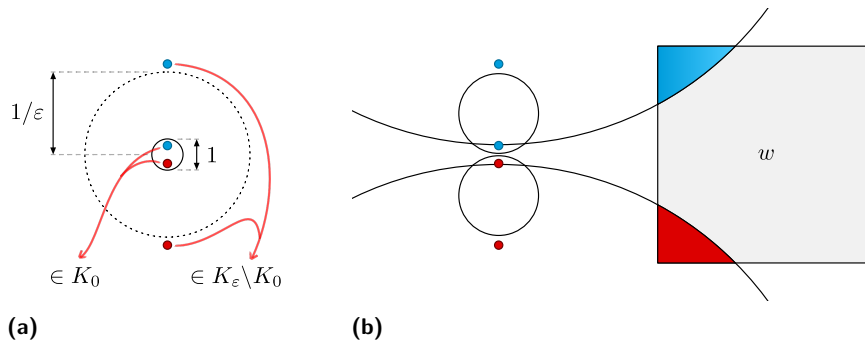
This leads to an initial approach to simplify an AVD. We distinguish between two types of leaf cells, based on the points inside their corresponding ε -representative sets. Any leaf cell w is said to be:

- **Resolved:** If every point in R_w belongs to the same class.
- **Ambiguous:** Otherwise, if at least two points in R_w belong to different classes.

Clearly, there is no need to store the set of ε -representatives of any resolved leaf cell, as instead, we can simply mark the leaf cell w as *resolved with the class* that is shared by all the points in R_w . This effectively reduces the space needed for such cells to be constant.

Furthermore, it seems that the bulk of the “work” needed to decide the class of a given query point can be carried out by the ambiguous leaf cells, along with some groupings of resolved leaf cells. The data structure presented in this paper, called Chromatic AVD, builds upon this hypothesis.

Additionally, we formally define the set of ε -border points of the training set P . This set, denoted as K_ε , contains any point $p \in P$ for which there exist some $q \in \mathbb{R}^d$ and $\bar{p} \in P$, such that p and \bar{p} are ε -approximate nearest-neighbors of q , and both belong to different classes. Denote $k_\varepsilon = |K_\varepsilon|$ as the number of ε -border points of the training set P . Note that $K_\varepsilon \subseteq K_{\varepsilon'}$ if and only if $\varepsilon \leq \varepsilon'$. Additionally, note that K_0 defines the set of (exact) border points of P , where $k = k_0$.



■ **Figure 2** Intuition to think that K_ε (and not K_0) is needed to ε -classify some query points.

This generalization of the definition of border points seems better suited to analyze the problem of ε -classification, as illustrated in Figure 2. Figure 2b shows the ε -approximate bisectors between the two closest and two farthest points (the first two belong to K_0 , while the others belong to K_ε but not K_0). A hypothetical leaf cell w is sufficiently separated from the only two exact border points, but intersects the ε -approximate bisectors between the

two farthest points. This implies that inside the cell w lie query points that *can only* be ε -classified with one class, and others with the other class, forcing this cell to be ambiguous. This suggests that K_0 is insufficient to account for the necessary complexity of ε -classification.

3 Chromatic AVD Construction

In this section, we describe our method for constructing the proposed Chromatic AVD. The following overview outlines the necessary steps followed to construct this data structure.

- *The Build step* (Section 3.1): Consists of building an initial quadtree-based subdivision of space, designed specifically to achieve the properties described in Lemma 3.
- *The Reduce step* (Section 3.2): Seeks to identify the leaf cells of the initial subdivision that are relevant for ε -classification, as well as those that can be ignored or simplified. This process consists of the following substeps.
 - Computing the sets of ε -representatives for every leaf cell of the initial quadtree.
 - Based on these sets, marking the leaf cells as either ambiguous or resolved.
 - Selecting those leaf cells which are relevant for ε -classification.
 - Building a new quadtree-based subdivision using the previously selected leaf cells.

3.1 The Build Step

We begin by constructing the tree T_{init} using similar methods as the ones used to construct a standard AVD. Thus, the first step is to compute a well-separated pair decomposition \mathcal{D} of P using a constant separation factor of $\sigma > 4$. While the standard construction would use all pairs in this decomposition, for the purpose of the Chromatic AVD, we filter \mathcal{D} to only keep bichromatic pairs. Denote $\mathcal{D}' \subseteq \mathcal{D}$ to be the set of bichromatic pairs in \mathcal{D} , where a pair $\mathcal{P} \in \mathcal{D}$ is said to be bichromatic if and only if the dumbbell heads separate points of different classes. Note that \mathcal{D}' can be computed similarly to \mathcal{D} , using a simple modification of the well-known algorithm for computing WSPDs [10] (the details are left to the reader).

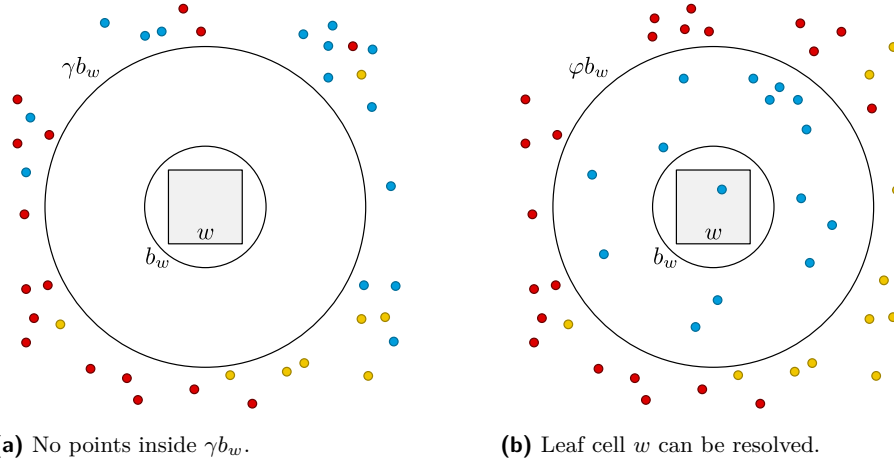
Next, we compute an initial set of quadtree boxes $\mathcal{U}(\mathcal{P})$ for every pair in \mathcal{D}' as follows. This construction depends on two constants c_1 and c_2 whose assignment will be described later in this section. For $0 \leq i \leq \lceil \log(c_1 1/\varepsilon) \rceil$, we define $b_i(\mathcal{P})$ as the ball centered at z of radius $r_i = 2^i \ell$. Thus, this set of balls involves radius values ranging from ℓ to $\Theta(\ell/\varepsilon)$. For each such ball $b_i(\mathcal{P})$, let $\mathcal{U}_i(\mathcal{P})$ be the set of quadtree boxes of size $r_i/(c_2 \gamma)$ that overlap the ball. Let $\mathcal{U}(\mathcal{P})$ denote the union of all these boxes over all the $\mathcal{O}(\log 1/\varepsilon)$ values of i .

After performing this process on every pair of the filtered decomposition \mathcal{D}' , take the union of all these boxes denoted as $\mathcal{U} = \bigcup_{\mathcal{P} \in \mathcal{D}'} \mathcal{U}(\mathcal{P})$. Finally, build the tree T_{init} from the set of quadtree boxes \mathcal{U} , leveraging property 2 of quadtrees described in Section 2. Additionally, for each class i in the training set, build an auxiliary tree T_{aux}^i from the point set P_i (i.e., the points of P of that are labeled with class i), using property 1 of quadtrees. These auxiliary trees will be used together with T_{init} in order to build our final tree T , the Chromatic AVD.

While the standard AVD construction satisfies that all resulting leaf cells of the tree have certain separation properties from the points of set P , the same is not true for tree T_{init} . However, the following result describes a relaxed notion of the separation properties, now based on the classes of the points, which are achieved by T_{init} .

► **Lemma 3** (Chromatic Separation Properties). *Given two separation parameters $\gamma > 2$ and $\varphi > 3$, every leaf cell w of the tree T_{init} satisfies at least one of the following separation properties, where b_w is the minimum enclosing ball of w :*

- (i) $P \cap \gamma b_w$ is empty (see Figure 3a), and hence b_w is concentrically γ -separated from P .
- (ii) The cell w can be resolved with the classes present inside $P \cap \varphi b_w$ (see Figure 3b).



■ **Figure 3** Basic separation properties achieved by during the construction of the Chromatic AVD.

Proof. Let w be any leaf cell of T_{init} , with center c_w and side length s_w , where its (minimum) enclosing ball b_w has radius $r_w = \sqrt{d}/2 s_w$ and shares the center c_w . Additionally, let $x_i \in P_i$ be a 1-approximate nearest-neighbor of c_w among the points of P of class i . In other words, for each class of points we use the auxiliary trees T_{aux}^i to compute a 1-approximate nearest-neighbor of c_w . A few cases unfold from here:

The first case is rather simple. If $4\gamma\varphi b_w \cap \{x_i\}_i = \emptyset$, knowing that the points x_i are 1-approximate nearest-neighbors of c_w , this implies that the ball $2\gamma\varphi b_w$ is empty (*i.e.*, we know that $2\gamma\varphi b_w \cap P = \emptyset$). Clearly, this means the first separation property holds for w .

Consider the case when $|4\gamma\varphi b_w \cap \{x_i\}_i| = 1$, and let i be the class of the point that lies inside $4\gamma\varphi b_w$. Following similar arguments to the previous case, this implies that only points of class i could potentially lie inside of $2\gamma\varphi b_w$. Then, check if x_i lies inside the smaller ball expansion $2\gamma b_w$. If not, we know that γb_w is empty (*i.e.*, $\gamma b_w \cap P = \emptyset$), making the first separation property hold for w . Otherwise, we know that $2\gamma b_w$ contains at least one point (*i.e.*, x_i), and additionally we know that $2\gamma b_w$ is φ -separated from points of all other classes but i (as $2\gamma\varphi b_w$ only contains points of class i). Given that $\varphi > 3$, the nearest neighbor of every query point inside $2\gamma b_w$ has class i . Therefore, w can be resolved with class i (namely, $C_w = \{i\}$), satisfying the second separation property.

Lastly, it is possible that $|4\gamma\varphi b_w \cap \{x_i\}_i| \geq 2$. However, it is possible to show that if this is the case, it immediately implies that every point inside $4\gamma\varphi b_w$ actually lies inside of some ball b'_w which is β -separated from w (see Figure 5a), where $\beta = 1/\varepsilon$. The details of this part of the proof are omitted, and left in the Appendix A, as the arguments are similar to the ones described in [5]. However, proving this provides insights into how to set constants c_1 and c_2 , where $c_1 \geq 3(1 + \varepsilon)$ and $c_2 \geq 20\varphi\sqrt{d}$.

Finally, if $|4\gamma\varphi b_w \cap \{x_i\}_i| \geq 2$, we know all points inside $4\gamma\varphi b_w$ are β -separated from w . We can now proceed similarly to the previous case, by checking if one of the computed 1-approximate nearest-neighbors lies inside the ball $2\gamma b_w$. If $2\gamma b_w \cap \{x_i\}_i = \emptyset$ we know that γb_w is empty (*i.e.*, $\gamma b_w \cap P = \emptyset$), making the first separation property hold for w . Otherwise, note that b'_w is completely contained inside $2\gamma(1 + \varepsilon)b_w$. Given that $\varphi > 3$, it is possible to show that for any query point in w , all points in b'_w are valid ε -approximate nearest neighbors. This implies that we can resolve w with the class of any of the points inside of b'_w , thus satisfying the second separation property. In particular, we mark w as resolved with every class present in the inner cluster b'_w , namely, $C_w = \{l(p) \mid \forall p \in b'_w \cap P\} = \{i \mid x_i \in 4\gamma\varphi b_w\}$. ◀

3.2 The Reduce Step

From the initial partitioning as described in Lemma 3, every leaf cell w of T_{init} is either concentrically γ -separated from P (i.e., $\gamma b_w \cap P = \emptyset$), or it is already marked as resolved. For every leaf cell w in the first case, we will compute a set of ε -representatives by leveraging the *concentric ball lemma* (see Lemma 5.1 in [5]). It states that there exists a set R_w of ε -representatives for w of size $\mathcal{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$, and provides a way to compute such set.

Instead of directly applying this result, we use it to compute a set of $\varepsilon/3$ -representatives for any leaf cell w that is yet unresolved. Essentially, this leads to the same asymptotic upper-bound on the size of R_w , meaning that $|R_w| = \mathcal{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$. Once R_w is computed, we can proceed to mark w as either resolved or ambiguous as follows.

Procedure to Mark Leaf Cells. For every leaf cell w , this procedure marks w as either resolved or ambiguous, following a few defined cases that unfold from the contents of the set R_w of points selected as representatives for w . Let $r_w^- = \varepsilon(1-\gamma)r_w/3$.

1. If all the points in R_w belong to the same class.

For every point $p \in R_w$ and class $i \in C$, compute a 1-approximate nearest-neighbor of p among the points of P_i , denoted as the point $x_{p,i}$. If $d(p, x_{p,i}) < r_w^-$, then add $x_{p,i}$ to R_w . It is easy to show that $x_{p,i}$ would also be an ε -representative for w . Repeat this for every point originally in R_w , and every class in the training set.

 - a. If any point $x_{p,i}$ was added to R_w , proceed with *Case 2*.
 - b. Otherwise, mark w as resolved with the class of the points in R_w . Namely, let i be the class of every point in R_w , then $C_w = \{i\}$.
2. If R_w contains points of more than one class.

Before proceeding, we will do some basic pruning of the set R_w . For every class i , compute a *net* among the points of R_w of class i , using a radius of r_w^- to compute the net, and replace the points of class i in R_w with the computed net. It is easy to see that the remaining points of R_w are a set of ε -representatives of w , and that every two points in R_w of the same class are at distance at least r_w^- .

 - a. If the diameter of R_w is less than r_w^- , it is easy to prove that all the points in R_w are ε -representatives of any point inside b_w . Therefore, w can be labeled as resolved with the class of all of the points in R_w . That is, $C_w = \{l(p) \mid \forall p \in R_w\}$.
 - b. If the diameter is greater than or equal to r_w^- , w is marked as ambiguous.

Let \mathcal{A} and \mathcal{R} be the sets of ambiguous and resolved leaf cells of T_{init} , respectively. We will use some of these cells to build the Chromatic AVD, while ignoring the remaining cells.

Consider the set of resolved leaf cells \mathcal{R} , we partition this set into two subsets \mathcal{R}_b and \mathcal{R}_i (named *boundary* and *interior* resolved leaf cells, respectively). We say a resolved leaf cell w_1 belongs to \mathcal{R}_b , if and only if there exists another resolved leaf cell w_2 adjacent to w_1 , such that $C_{w_2} \setminus C_{w_1} \neq \emptyset$. Every other resolved leaf cell belongs to \mathcal{R}_i (i.e., $\mathcal{R}_i = \mathcal{R} \setminus \mathcal{R}_b$). Note that both sets \mathcal{R}_b and \mathcal{R}_i can be identified by a simple traversal over the leaf cells of T_{init} , using linear time in the size of the tree².

Finally, we build a new tree T from the set of ambiguous and boundary resolved leaf cells $\mathcal{A} \cup \mathcal{R}_b$. By well-known construction methods of quadrees, as described in Section 2, the leaf cells of T either belong to $\mathcal{A} \cup \mathcal{R}_b$, or are “Steiner” leaf cells added during the construction of T that cover the remainder of the space that is uncovered by $\mathcal{A} \cup \mathcal{R}_b$.

² Two leaf cells are adjacent if and only if a vertex of one of the cells “touches” the other cell. This implies that the number of adjacency relations (i.e., edges in the implicit graph where the leaf cells are the nodes) is $\mathcal{O}(2^d m)$, where m is the number of leaf cells of the tree T_{init} .

► **Lemma 4.** *For any leaf cell w in the tree T such that $w \notin \mathcal{A} \cup \mathcal{R}_b$, w must cover a space that is also covered by a collection of leaf cells of T_{init} , all of which are resolved with the same set of classes C_w .*

Proof. This becomes apparent from the construction of T . In the new tree T , consider any leaf cell w of T that is not part of $\mathcal{A} \cup \mathcal{R}_b$ (i.e., a “Steiner” leaf cell added during the construction of the tree). Now, recall that the leaf cells of both T and T_{init} are a partitioning of (the same) space, which means that we can define $\mathcal{W}_w = \{w' \in T_{\text{init}} \mid w \cap w' \neq \emptyset\}$ as the collection of leaf cells of T_{init} that cover the same space covered by w .

Now, for any fixed w of T , it is easy to see that any two leaf cells $w_1, w_2 \in \mathcal{W}_w$ must be resolved with the same set of classes. Otherwise, at least one of these two would be part of the set \mathcal{R}_b , which would be a contradiction to the fact that w is a “Steiner” leaf cell of T . Therefore, any query inside w can be answered with the classes $C_w = C_{w_1} = C_{w_2}$, and this can be determined during preprocessing by a single query on T_{init} (e.g., finding the leaf cell of T_{init} that contains the center c_w of w is sufficient to know the contents of C_w). ◀

This implies that after building tree T , and with some extra preprocessing to resolve the “Steiner” leaf cells of the tree, we can use the resulting data structure to correctly answer chromatic ε -approximate nearest-neighbor queries over the training set P . In other words, T can be used to answer ε -classification queries over P . We call this data structure T the Chromatic AVD.

► **Lemma 5.** *The construction of T takes $\tilde{O}(n\gamma^d \log \frac{1}{\varepsilon})$ time.*

Proof. Let’s analyze the total time needed to build our Chromatic AVD, namely the tree T , by analyzing the time required to perform each step of the construction.

- Building T_{init} has essentially the same complexity of building any standard AVD [3–5, 23]. This means that constructing T_{init} takes $\mathcal{O}(m \log m)$ time, where $m = n\gamma^d \log \frac{1}{\varepsilon}$. Note that during the construction, while computing the set of ε -representatives of each leaf cell, each leaf cell can already be marked as either ambiguous or resolved.
- Building the auxiliary trees T_{aux}^i for every class i , takes $\mathcal{O}(n \log n)$ time, as the number of classes of P is considered to be a constant. Recall that because these trees are only used to for 1-ANN queries, they only need to be standard Quadrees, and not AVDs.
- Identifying the set \mathcal{R}_b requires a traversal over the leaf-level partitioning of the space, which is linear in terms of the number of cells. Therefore, this step requires $\mathcal{O}(m)$ time.
- Once the sets of ambiguous and boundary resolved leaf cells are identified, namely, the sets \mathcal{A} and \mathcal{R}_b , the final tree T can be built. Roughly, this step takes $\mathcal{O}(m \log m)$ time.
- Finally, we must resolve the “Steiner” leaf cells of T , which can be done by a single query over T_{init} , each taking $\mathcal{O}(\log m)$ time. Thus, this step takes $\mathcal{O}(m \log m)$ total time.

All together, the total construction time is dominated by the first step. Therefore, the time required to construct T is $\mathcal{O}(m \log m) = \tilde{O}(m) = \tilde{O}(n\gamma^d \log \frac{1}{\varepsilon})$. ◀

4 Tree-size Analysis

4.1 Initial Size Bounds

Define the set of important leaf cells \mathcal{I} of the tree T_{init} as those leaf cells w for which there exists two ε -border points inside $\rho\gamma b_w$ for some constant ρ , such that the distance between these points is lower-bounded by $\Omega(\varepsilon\gamma r_w)$. Formally, we define this set as $\mathcal{I} = \{w \in T_{\text{init}} \mid \exists p_1, p_2 \in \rho\gamma b_w \cap K_\varepsilon, d(p_1, p_2) = \Omega(\varepsilon\gamma r_w)\}$.

► **Lemma 6.** *The number of important leaf cells of T_{init} is $|\mathcal{I}| = \mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$.*

Proof. This proof follows from a charging argument on the set K_ε of ε -border points of P . More specifically, consider a well-separated pair decomposition \mathcal{D}'' of K_ε with constant separation factor of $\sigma > 4$, the charging scheme assigns every important leaf cell $w \in \mathcal{I}$ to a pair of \mathcal{D}'' . Recall that \mathcal{D}'' can generally be considered to have $\mathcal{O}(k_\varepsilon)$ pairs, where $k_\varepsilon = |K_\varepsilon|$. It is important to note that both K_ε and \mathcal{D}'' need not be computed.

Given that $w \in \mathcal{I}$, we know there exist two points $p_1, p_2 \in \rho\gamma b_w \cap K_\varepsilon$. Let $\mathcal{P} \in \mathcal{D}''$ be the pair of \mathcal{D}'' that contains both p_1 and p_2 , each in one of its dumbbell heads. We then charge w to the pair \mathcal{P} . Denote z and ℓ to be the center and length of \mathcal{P} , respectively, we know the following. First, note that the distance from c_w (the center of w) to z is $d(c_w, z) \leq \rho\gamma r_w + \ell$. Additionally, we know that $\ell \geq d(p_1, p_2)/2 = \Omega(\varepsilon\gamma r_w)$ by the properties of WSPDs described in Section 2. Therefore, this implies that the ratio $d(c_w, z)/\ell = \mathcal{O}(1/\varepsilon)$.

Finally, fix some pair $\mathcal{P} \in \mathcal{D}''$ with center z and length ℓ , and consider all important leaf cells according to their distance to z . For any value of $i \in [0, 1, \dots, \mathcal{O}(\log 1/\varepsilon)]$, consider all leaf cells that can charge \mathcal{P} whose distance to z is between $2^i \ell$ and $2^{i+1} \ell$. From our previous analysis, $r_w \geq d(c_w, z)/\rho\gamma \geq 2^i \ell/\rho\gamma$. By a simple packing argument, the number of such leaf cells is at most $\mathcal{O}(\gamma^d)$. Thus, a total of $\mathcal{O}(\gamma^d \log 1/\varepsilon)$ cells can charge \mathcal{P} . Note that no leaf cell whose distance to z is $\Omega(\ell/\varepsilon)$ can charge \mathcal{P} , as it would contradict the fact that both p_1 and p_2 are separated by a distance of $\Omega(\varepsilon\gamma r_w)$. Finally, the proof follows by knowing that there are at most $\mathcal{O}(k_\varepsilon)$ pairs in \mathcal{D}'' . ◀

► **Lemma 7.** *Every ambiguous leaf cell of T_{init} is important, namely $\mathcal{A} \subseteq \mathcal{I}$.*

Proof. Consider any ambiguous leaf cell $w \in \mathcal{A}$ of the tree T_{init} . Knowing that w is ambiguous implies that there must exist some point $q \in \frac{\gamma}{2}b_w$ for which two of the ε -representatives of w are valid ε -approximate nearest neighbors for q , both points belong to different classes, and the distance between them is $\Omega(\varepsilon\gamma r_w)$. Formally, denote these points as $p_1, p_2 \in P$ such that $l(p_1) \neq l(p_2)$, $d(p_1, p_2) \geq \varepsilon(1-\gamma)r_w/4$, and $d(q, p_1), d(q, p_2) \leq (1+\varepsilon)d_{\text{nn}}(q)$.

We will see now how to bound the distance from c_w to any of these points as a constant factor of r_w (recall that $r_w = \sqrt{d}/2 s_w$). From the proof of Lemma 6.3 in [5], we know that the ball $c_3\gamma b_w \cap P \neq \emptyset$, for some constant $c_3 \geq 1 + 2c_2/\sqrt{d}$. In other words, $d_{\text{nn}}(c_w) \leq c_3\gamma r_w$. From this, we can say that $d_{\text{nn}}(q) \leq (\frac{1}{2} + c_3)\gamma r_w$. Applying the triangle inequality yields that $d(c_w, p_1) \leq d(c_w, q) + d(q, p_1) \leq (\frac{1}{2} + (1+\varepsilon)(\frac{1}{2} + c_3))\gamma r_w$. Similarly, we can achieve the same bound for $d(c_w, p_2)$.

Therefore, both $p_1, p_2 \in \rho\gamma b_w$ for sufficiently large constant ρ (i.e., $\rho \geq \varepsilon(\frac{1}{2} + c_3) + c_3 + 1$). Knowing also that $d(p_1, p_2) = \Omega(r_w^-) = \Omega(\varepsilon\gamma r_w)$ yields that the leaf cell $w \in \mathcal{I}$. ◀

► **Lemma 8.** *Every boundary resolved leaf cell of T_{init} is important, namely $\mathcal{R}_b \subseteq \mathcal{I}$.*

Proof. Let $w_1 \in \mathcal{R}_b$ be any boundary resolved leaf cell of the tree T_{init} , we know there exists another leaf cell $w_2 \in \mathcal{R}_b$ adjacent to w_1 , such that there exists some class $i \in C_{w_2} \setminus C_{w_1}$. Let b_{w_1} and b_{w_2} be the corresponding bounding balls of w_1 and w_2 . By definition, any point q on the boundary shared by w_1 and w_2 has at least one ε -representative from each cell, namely some points $p_1 \in R_{w_1}$ and $p_2 \in R_{w_2}$, where $l(p_1) \neq i$ and $l(p_2) = i$. Additionally, by similar arguments to the ones described in Lemma 7, we know that both $p_1, p_2 \in \rho\gamma b_w$ for sufficiently large constant ρ .

Now, we proceed to prove that $d(p_1, p_2) \geq r_w^-/2$. First, note that if w_1 was resolved by the initial marking of leaf cells as described in Lemma 3, then p_2 must lie outside of γb_w . In such cases, clearly $d(p_1, p_2) \geq r_w^-/2$. The remaining possibility is that w_1 was resolved after computing the set of representatives. From the described procedure, in Case 1, we know

that if $d(p_1, p_2) < r_w^-/2$, the point $x_{p_1, i}$ (which is a 1-approximate nearest-neighbor of p_1 among points in P_i) would hold that $d(p_1, x_{p_1, i}) < r_w^-$. Hence, $x_{p_1, i}$ should have been added to the set of representatives of w_1 , contradicting the assumption that w_1 is resolved, or that C_{w_1} does not contain i . All together, we have that $d(p_1, p_2) = \Omega(r_w^-) = \Omega(\varepsilon \gamma r_w)$. From the definition of the set of important leaf cells, $w \in \mathcal{I}$. \blacktriangleleft

Lemmas 7 and 8 imply that all the leaf cells used to build T belong to the set of important leaf cells (i.e., $\mathcal{A} \cup \mathcal{R}_b \subseteq \mathcal{I}$), whose size is upper-bounded by Lemma 6. All together, and leveraging construction methods of quadrees (see Section 2), the size of T is proportional to the total number of leaf cells used to build it, which we now know is $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$. However, we also need to account for the set of ε -representatives stored for each ambiguous leaf cell, leading to a worst-case upper-bound of $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon} \cdot 1/(\varepsilon \gamma)^{\frac{d-1}{2}})$ total space to store T .

4.2 Spatial Amortization

The previous result can be improved by applying a technique called *spatial amortization*, described by Arya *et al.* [5]. That is, we can remove the extra $\mathcal{O}(1/(\varepsilon \gamma)^{\frac{d-1}{2}})$ factor from the analysis of the space requirements for T .

This will be twofold process, as in order to successfully apply spatial amortization to the analysis of the data structure, we first need to further reduce the set of ε -representatives of every ambiguous leaf cell in the tree. Actually, the new set will no longer be a set of ε -representatives, but it will just be a *weak* ε -coreset for query points inside of each leaf cell.

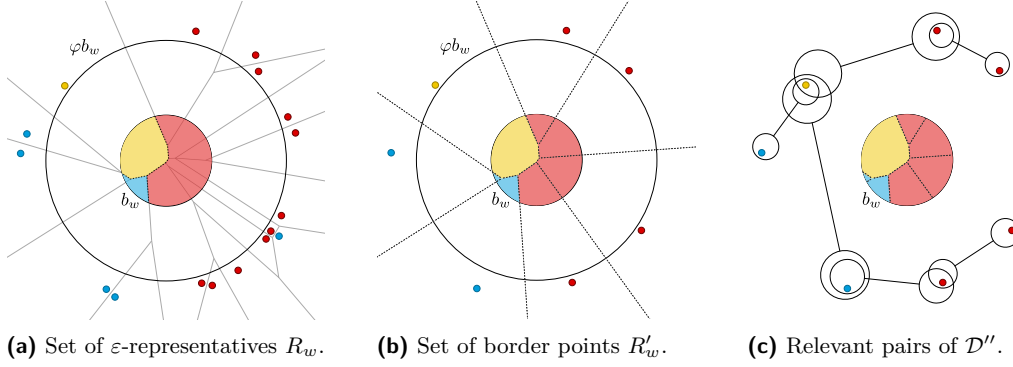
► **Lemma 9.** *The total space required to store the ambiguous leaf cells of T is $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$.*

Consider any ambiguous leaf cell w of T , and in particular, consider the set R_w of ε -representatives of w . By construction, R_w has the property that every point $q \in b_w$ has at least one ε -approximate nearest-neighbor in the set R_w . However, note that the opposite is not necessarily true, as not every $p \in R_w$ is an ε -approximate nearest-neighbor of some point in b_w . Even worst, while the fact the w is ambiguous indicates that at least two points in R_w belong to K_ε , the remaining points of R_w might not, which in turn prevents the application of a spatial amortization analysis. Overall, this suggests some of the points of R_w might not be necessary to distinguish between the classes that change the classification of points inside b_w (see Figure 4a).

This can be resolved as follows. Suppose we have access to the Voronoi diagram of the set of points R_w , and consider the boundaries between adjacent cells of this diagram. Any boundary that separates two points of R_w of different classes, and that intersects b_w , is relevant to the classification any query point inside b_w . Formally, we define the set $R'_w \subseteq R_w$ of border points of R_w as (see Figure 4b):

$$R'_w = \{p \in R_w \mid \exists q \in b_w, p' \in R_w \text{ such that } l(p) \neq l(p') \wedge d(q, p) = d(q, p')\}$$

This new set R'_w has some useful properties. Note that for any query point $q \in b_w$, its (exact) nearest-neighbor in R'_w belongs to the same class as its (exact) nearest-neighbor in R_w , which itself is an ε -approximate nearest-neighbor of q among the points of P . In other words, R'_w is an ε -coreset for any query point in b_w . This implies that we can replace the set of ε -representatives of w with the set R'_w . Moreover, this means that by the definition of ε -border points, $R'_w \subseteq K_\varepsilon$. Note that we don't need to compute the Voronoi diagram of R_w , but instead just part of the 1-skeleton. While not immediately evident, the set R'_w can be computed in time $\mathcal{O}(|R_w|^3)$, by fixing every two pairs of R_w , and checking whether there exists a point $q \in b_w$ with the desired properties. The later step can be solved using Linear Programming via the lifting transform into a parabola in $d + 1$ Euclidean space.



■ **Figure 4** The set R_w of ε -representatives of w can be reduced to the set R'_w . This latter set is a subset of K_ε , and can be charged to a proportional number of relevant pairs of \mathcal{D}'' .

Now, let's proceed with the charging argument over the pairs of the same WSPD \mathcal{D}'' used in Lemma 6. Instead of only charging w to a single pair (as described in Lemma 7), we charge every point stored in R'_w to a pair of \mathcal{D}'' . Thus, consider the following procedure to find a sufficient number of pairs to charge all the points in R'_w , which is derived from a similar procedure proposed in [5]. See Figure 4c for an illustrative example.

1. Compute a net of R'_w using radius r_w^- , and denote this subset R''_w . Given that all the points of R'_w that belong to the same class are already separated by a distance of at least r_w^- , we know that $|R''_w| = \Theta(|R'_w|)$, hiding constants³ that depend exponentially on d .
2. Find the two points of $p_1, p_2 \in R''_w$ with smallest pairwise distance, and consider the pair of $\mathcal{P} \in \mathcal{D}''$ that contains both points p_1 and p_2 , each in one of its dumbbell heads. Note that by having computed a net in the previous step, $d(p_1, p_2) \geq r_w^-$.
3. Delete one of the two points from R''_w (without loss of generality, delete p_1).
4. Charge every point of R'_w that is covered by p_1 (i.e., whose distance to p_1 is $\leq r_w^-$) to the pair \mathcal{P} . By the arguments described in step 1 on the size of R''_w , we know that \mathcal{P} receives a charge from $\mathcal{O}(1)$ points of R'_w .
5. Repeat steps 2-4 with the remaining points of R''_w until the set is empty.

Evidently, the number of pairs found (and charged) equals $|R''_w| - 1$. All together, we have that the total space required to store all the ambiguous leaf cells is proportional to the sum of charges to every pair of \mathcal{D}'' . Using the same arguments as Lemma 6, this implies that the total storage is $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$. This completes the proof of Theorem 1.

References

- 1 Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 2018.
- 2 Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.
- 3 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Optimal approximate polytope membership. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–288. SIAM, 2017.

³ More specifically, we know that $|R''_w| \geq |R'_w| / \phi^{d-1} c$, where ϕ^{d-1} is the kissing number in d -dimensional Euclidean space, and c is the number of classes in P .

- 4 Sunil Arya, Guilherme D. Da Fonseca, and David M. Mount. Approximate polytope membership queries. *SIAM Journal on Computing*, 47(1):1–51, 2018.
- 5 Sunil Arya, Theodoros Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.
- 6 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998. doi:10.1145/293347.293348.
- 7 Ricardo Barandela, Francesc J Ferri, and J Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.
- 8 Ahmad Biniiaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel Smid. On the minimum consistent subset problem. In *Workshop on Algorithms and Data Structures*, pages 155–167. Springer, 2019.
- 9 Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.
- 10 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.
- 11 Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.
- 12 T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, January 1967. doi:10.1109/TIT.1967.1053964.
- 13 Luc Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):75–78, 1981.
- 14 Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
- 15 E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, 1951.
- 16 Alejandro Flores-Velazco. Social distancing is good for points too! In *Proceedings of the 32st Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada, 2020*.
- 17 Alejandro Flores-Velazco and David M. Mount. Guarantees on nearest-neighbor condensation heuristics. In *Proceedings of the 31st Canadian Conference on Computational Geometry, CCCG 2019, August 8-10, 2019, University of Alberta, Edmonton, Alberta, Canada, 2019*.
- 18 Alejandro Flores-Velazco and David M. Mount. Coresets for the nearest-neighbor rule, 2020. arXiv:2002.06650.
- 19 Alejandro Flores-Velazco and David M. Mount. Coresets for the Nearest-Neighbor Rule. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.47.
- 20 Alejandro Flores-Velazco and David M. Mount. Guarantees on nearest-neighbor condensation heuristics. *Computational Geometry*, 95:101732, 2021. doi:10.1016/j.comgeo.2020.101732.
- 21 Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, pages 370–378, 2014.
- 22 Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020. URL: <https://arxiv.org/abs/1908.10396>.
- 23 Sarel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

- 24 P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, 1968. doi:10.1109/TIT.1968.1054155.
- 25 Norbert Jankowski and Marek Grochowski. Comparison of instances selection algorithms I. Algorithms survey. In *Artificial Intelligence and Soft Computing-ICAISC 2004*, pages 598–603. Springer, 2004.
- 26 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017.
- 27 Marc Khoury and Dylan Hadfield-Menell. Adversarial training with Voronoi constraints. *CoRR*, abs/1905.01019, 2019. arXiv:1905.01019.
- 28 Nicolas Papernot and Patrick McDaniel. Deep k -nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.
- 29 Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. Deep k -nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.
- 30 G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.
- 31 Hanan Samet. *The design and analysis of spatial data structures*, volume 85. Addison-Wesley Reading, MA, 1990.
- 32 Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. arXiv:1404.7828.
- 33 Chawin Sitawarin and David Wagner. On the robustness of deep k -nearest neighbors. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2019.
- 34 Charles J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.
- 35 Godfried Toussaint. Open problems in geometric methods for instance-based learning. In Jin Akiyama and Mikio Kano, editors, *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2002. doi:10.1007/978-3-540-44400-8_29.

A Proof Details

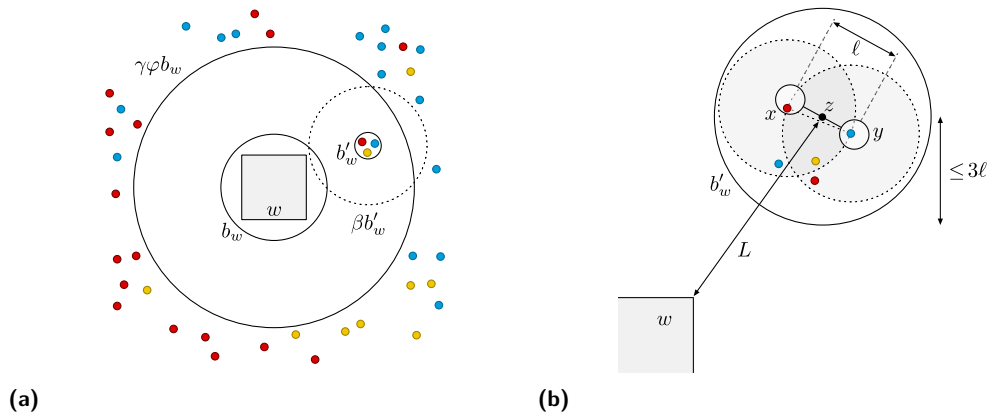


Figure 5 It is possible that points of ≥ 2 classes lie inside of $\gamma\varphi b_w$. However, this case can be reduced to the two separation properties illustrated in Figure 3.

To finish the proof of Lemma 3, we shall proof the assumption that if $|4\gamma\varphi b_w \cap \{x_i\}_i| \geq 2$, then every point inside $4\gamma\varphi b_w$ lies inside the ball b'_w which is β -separated from w .

Proof. Let $x, y \in 4\gamma\varphi b_w$ be the two points of different classes inside the ball $4\gamma\varphi b_w$ with largest pairwise distance. Thus, it is easy to show that all the points inside $4\gamma\varphi b_w$ lie inside the two balls centered at x and y with radii equal to $d(x, y)$, as shown in Figure 5b. By definition of the (bichromatic) well-separated pair decomposition \mathcal{D}' , there exists a pair $\mathcal{P} \in \mathcal{D}'$ that contains x and y each in one of its dumbbells, with length ℓ and center z . Now, we define the ball b'_w with center at z and radius $r'_w = \max(d(z, x), d(z, y)) + d(x, y)$. By definition of \mathcal{P} , we know that $d(z, x), d(z, y) \leq \ell$ and $d(x, y) \leq 2\ell$, thus making $r'_w \leq 3\ell$. Let L be the distance from w to z , we distinguish two cases based on the relationship between L and ℓ :

- $L > c_1\beta\ell$. Consider the distance that separates the ball b'_w from the cell w .

$$d(w, b'_w) = L - r'_w > c_1\beta\ell - r'_w \geq (c_1\beta/3 - 1)r'_w$$

Since $\beta = 1/\varepsilon$, for all sufficiently large constants $c_1 \geq 3(1 + \varepsilon)$, the distance $d(w, b'_w)$ exceeds $\beta r'_w$ which implies that b'_w is concentrically β -separated from w .

- $L \leq c_1\beta\ell$. We will show that this case cannot occur, since otherwise the dumbbell \mathcal{P} would have caused w to be split, contradicting the assumption that it is a leaf cell of T_{init} . Since x, y , and w are all contained in the ball $4\gamma\varphi b_w$ whose center lies within w , we have both that $d(x, w) \leq 4\gamma\varphi r_w$, and $\ell < 2d(x, y) \leq 2(8\gamma\varphi r_w) = 16\gamma\varphi r_w$. Thus, by the triangle inequality, we have:

$$L \leq d(x, y) + d(x, w) < \ell + 4\gamma\varphi r_w < 16\gamma\varphi r_w + 4\gamma\varphi r_w = 20\gamma\varphi r_w$$

Because $L \leq c_1\beta\ell$, it follows from our construction that there is at least one ball $b_i(\mathcal{P})$ (with $0 \leq i \leq \lceil \log c_1\beta \rceil$) that overlaps w . Let b denote the smallest such ball, and let r denote its radius. By the construction, we have that $r \leq \max(\ell, 2L)$. Since our construction generates all quadtree boxes of size $r/(c_2\gamma)$ that overlap b , it follows that $s_w \leq r/(c_2\gamma)$. Thus, we have:

$$r_w = s_w \frac{\sqrt{d}}{2} \leq \frac{r\sqrt{d}}{2c_2\gamma} \leq \frac{\max(\ell, 2L)\sqrt{d}}{2c_2\gamma} < \frac{20\gamma\varphi r_w \sqrt{d}}{c_2\gamma} = \frac{20\varphi r_w \sqrt{d}}{c_2}$$

Choosing $c_2 \geq 20\varphi\sqrt{d}$ yields the desired contradiction.

This completes the proof of Lemma 3. ◀

Compression by Contracting Straight-Line Programs

Moses Ganardi 

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

In grammar-based compression a string is represented by a context-free grammar, also called a *straight-line program (SLP)*, that generates only that string. We refine a recent balancing result stating that one can transform an SLP of size g in linear time into an equivalent SLP of size $\mathcal{O}(g)$ so that the height of the unique derivation tree is $\mathcal{O}(\log N)$ where N is the length of the represented string (FOCS 2019). We introduce a new class of balanced SLPs, called *contracting* SLPs, where for every rule $A \rightarrow \beta_1 \dots \beta_k$ the string length of every variable β_i on the right-hand side is smaller by a constant factor than the string length of A . In particular, the derivation tree of a contracting SLP has the property that every subtree has logarithmic height in its leaf size. We show that a given SLP of size g can be transformed in linear time into an equivalent contracting SLP of size $\mathcal{O}(g)$ with rules of constant length. This result is complemented by a lower bound, proving that converting SLPs into so called α -balanced SLPs or AVL-grammars can incur an increase by a factor of $\Omega(\log N)$.

We present an application to the navigation problem in compressed unranked trees, represented by *forest straight-line programs (FSLPs)*. A linear space data structure by Reh and Sieber (2020) supports navigation steps such as going to the parent, left/right sibling, or to the first/last child in constant time. We extend their solution by the operation of moving to the i -th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node.

Contracting SLPs are also applied to the finger search problem over SLP-compressed strings where one wants to access positions near to a pre-specified *finger* position, ideally in $\mathcal{O}(\log d)$ time where d is the distance between the accessed position and the finger. We give a linear space solution for the dynamic variant where one can set the finger in $\mathcal{O}(\log N)$ time, and then access symbols or move the finger in time $\mathcal{O}(\log d + \log^{(t)} N)$ for any constant t where $\log^{(t)} N$ is the t -fold logarithm of N . This improves a previous solution by Bille, Christiansen, Cording, and Gørtz (2018) with access/move time $\mathcal{O}(\log d + \log \log N)$.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases grammar-based compression, balancing, finger search

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.45

Related Version *Full Version*: <https://arxiv.org/abs/2107.00446> [9]

Acknowledgements The author thanks Paweł Gawrychowski, Artur Jeż, Philipp Reh, and Louisa Seelbach Benkner for helpful discussions. The author is also indebted to the anonymous referees whose comments improved the presentation of this work.

1 Introduction

In grammar-based compression a long string is represented by a context-free grammar, also called a *straight-line program (SLP)*, that generates only that string. Straight-line programs can achieve exponential compression, e.g. a string of length 2^n can be produced by the grammar with the rules $A_n \rightarrow A_{n-1}A_{n-1}, \dots, A_0 \rightarrow a$. While it is NP-hard to compute a smallest SLP for a given string [5] there are efficient grammar-based compressors of both practical and theoretical interest such as the LZ78/LZW-algorithms [25, 24], SEQUITUR [19], and RE-PAIR [16]. There is a close connection between grammar-based compression and the LZ77 algorithm, which parses a string into z phrases (without self-references): On the one



© Moses Ganardi;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, 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

hand z is always a lower bound on the size of the smallest SLP for the string [5]. On the other hand one can always construct an SLP of size $\mathcal{O}(z \log N)$ where N is the string length [5, 22] (see also [13] for LZ77 with self-referential phrases). Furthermore, the hierarchical structure of straight-line programs makes them amenable to algorithms that work directly on the compressed representation, without decompressing the string first. We refer to [17] for a survey on the broad literature on algorithms on grammar-compressed data.

Balanced grammars. For some algorithmic applications it is useful if the SLP at hand satisfies certain balancedness conditions. In the following we always denote by N the length of the represented string. A recent result states that one can transform an SLP of size g in linear time into an equivalent SLP of size $\mathcal{O}(g)$ so that the height of the unique derivation tree is $\mathcal{O}(\log N)$ [10]. This yields a clean $\mathcal{O}(g)$ space data structure which supports random access to any position i in the string in time $\mathcal{O}(\log N)$, by descending in the derivation tree from the root to the i -th leaf. The original solution for the random access problem by Bille, Landau, Raman, Sadakane, Satti, and Weimann relied on a sophisticated weighted ancestor data structure [3]. Its advantage over the balancing approach from [10] is that it supports random access to the string defined by any given variable A in time $\mathcal{O}(\log |A|)$.

Although the derivation tree of an SLP may have logarithmic height its subtrees may still be very unbalanced. Arguably, the strongest balancedness notions are α -balanced SLPs [5] and AVL-grammars [22]. An SLP in Chomsky normal form is α -balanced if for every rule $A \rightarrow BC$ the ratios $|B|/|A|$ and $|C|/|A|$ lie between α and $1 - \alpha$. An *AVL-grammar* is an SLP in Chomsky normal form whose derivation tree is an AVL-tree, i.e. for every rule $A \rightarrow BC$ the subtree heights below B and C differ at most by one. In fact, the aforementioned transformations from LZ77 into SLPs produce an α -balanced SLP, with $\alpha \leq 1 - \frac{1}{2}\sqrt{2}$ [5], and an AVL-grammar [22]. Using the same proof techniques one can also transform an SLP of size g into an α -balanced SLP or an AVL-grammar of size $\mathcal{O}(g \log N)$ [5, 22].

Let us list a few algorithmic results on α -balanced SLPs and AVL-grammars. The *compressed pattern matching problem* can be solved in linear time if the text is given by an α -balanced SLP and the pattern is given explicitly [13]. Gagie, Gawrychowski, Kärkkäinen, Nekrich, and Puglisi [7] presented a solution for the *bookmarking problem* in α -balanced SLPs or AVL-grammars of size g . Given b positions in the string, called bookmarks, we can decompress any substring of length ℓ that covers a bookmark in time $\mathcal{O}(\ell)$ and space $\mathcal{O}(g + b \log^* N)$. Based on this bookmarking data structure they present *self-indexes* for LZ77 and SLPs [7, 8], which support extracting substrings and finding all occurrences of a given pattern. Abboud, Backurs, Bringmann, and Künnemann studied the Hamming distance problem and the subsequence problem on SLP-compressed strings [1]. As a first step their algorithms convert the input SLPs into AVL-grammars, and solve both problems in time $\tilde{\mathcal{O}}(g^{1.410} \cdot N^{0.593})$, improving on the decompress-and-solve $\mathcal{O}(N)$ time algorithms.

Main results. The starting point of this paper is the observation that the size increase by a $\mathcal{O}(\log N)$ factor in the transformation from SLPs to α -balanced SLPs or AVL-grammars is unavoidable (Theorem 4). This lower bound holds whenever in the derivation tree any path from a variable A to a leaf has length $\Theta(\log |A|)$. This motivates the search for balancedness notions of SLPs that can be established without increasing the size by more than a constant factor and that provide good algorithmic properties. We introduce a new class of balanced SLPs, called *contracting straight-line programs*, in which every variable β_i occurring on the right-hand side of a rule $A \rightarrow \beta_1 \dots \beta_k$ satisfies $|\beta_i| \leq |A|/2$. The derivation tree of an contracting SLP has the property that every subtree has logarithmic height in its leaf size,

i.e. in the number of descendant leaves. We explicitly admit rules with right-hand sides of length greater than two, however, the length will always be bounded by a constant in this paper. We say that an SLP \mathcal{G} defines a string s if some variable in \mathcal{G} derives s (and s only). The main theorem of this paper refines the balancing theorem from [10] as follows:

► **Theorem 1.** *Given an SLP \mathcal{G} of size g , one can compute in linear time a contracting SLP of size $\mathcal{O}(g)$ with constant-length right-hand sides which defines all strings that \mathcal{G} defines.*

As an immediate corollary we obtain a simple $\mathcal{O}(g)$ size data structure supporting access to the i -th symbol of a variable A in time $\mathcal{O}(\log |A|)$ instead of $\mathcal{O}(\log N)$. This is useful whenever multiple strings s_1, \dots, s_m are compressed using a single SLP since we can support random access to any string s_i in time $\mathcal{O}(\log |s_i|)$. We present an example application to unranked trees represented by *forest straight-line programs (FSLPs)*. FSLPs are a natural generalization of SLPs that can compress trees both horizontally and vertically, and share the good algorithmic applicability of their string counterparts [11]. Reh and Sieber presented a linear space data structure on FSLP-compressed trees that allows to perform various navigation steps in constant time [21]. We extend their data structure by the operation of moving to the i -th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node.

► **Theorem 2.** *Given an FSLP \mathcal{G} of size g , one can compute an data structure in $\mathcal{O}(g)$ time and space supporting the following operations in constant time: Move to the root of the first/last tree of a given variable, move to the first/last child, to the left/right sibling or to the parent of the current node, return the symbol of the current node. One can also move to the i -th child of the current node in time $\mathcal{O}(\log d)$ where d is the degree of the current node.*

A second application concerns the *finger search problem* on grammar-compressed strings. A finger search data structure supports fast updates and searches to elements that have small rank distance from the *fingers*, which are pointers to elements in the data structure. The survey [4] provides a good overview on dynamic finger search trees. In the setting of finger search on a string s , Bille, Christiansen, Cording, and Gørtz [2] considered three operations: `access(i)` returns symbol $s[i]$, `setfinger(i)` sets the finger at position i of s , and `movefinger(i)` moves the finger to position i in s . Given an SLP of size g for a string of length N , they presented an $\mathcal{O}(g)$ size data structure which supports `setfinger(i)` in $\mathcal{O}(\log N)$ time, and `access(i)` and `movefinger(i)` in $\mathcal{O}(\log d + \log \log N)$ time where d is the distance from the current finger position [2]. If we assume that the SLP is α -balanced or an AVL-grammar, there is a linear space solution supporting `access(i)` and `movefinger(i)` in $\mathcal{O}(\log d)$ time (Theorem 17). For general SLPs we present a finger search structure with improved time bounds:

► **Theorem 3.** *Let $t \geq 1$. Given an SLP of size g for a string of length N , one can support `setfinger(i)` in $\mathcal{O}(\log N)$ time, and `access(i)` and `movefinger(i)` in $\mathcal{O}(\log d + \log^{(t)} N)$ time, where d is the distance between i and the current finger position, after $\mathcal{O}(tg)$ preprocessing time and space.*

Here $\log^{(t)} N$ is the t -fold logarithm of N , i.e. $\log^{(0)} N = N$ and $\log^{(t+1)} N = \log \log^{(t)} N$. Choosing any constant t we obtain a linear space solution for dynamic finger search, supporting `access(i)` and `movefinger(i)` in $\mathcal{O}(\log d + \log^{(t)} N)$ time. Alternatively, we obtain a clean $\mathcal{O}(\log d)$ time solution if we admit a $\mathcal{O}(g \log^* N)$ space data structure. Theorem 3 also works for multiple fingers where every finger uses additional $\mathcal{O}(\log N)$ space.

Let us remark that Theorem 1 holds in the *pointer machine model* [23], whereas for Theorem 2 and Theorem 3 we assume the *word RAM model* with the standard arithmetic and bitwise operations on w -bit words, where $w \geq \log N$. The assumption on the word length is standard in the area of grammar-based compression, see [3, 2].

Overview of the proofs. The proof of Theorem 1 follows the ideas from [3] and [10]. The obstacle for $\mathcal{O}(\log N)$ time random access or $\mathcal{O}(\log N)$ height are occurrences of *heavy* variables on right-hand sides of rules $A \rightarrow \beta_1 \dots \beta_k$, i.e. variables β_i whose length exceeds $|A|/2$. These occurrences can be summarized in the *heavy forest*, which is a subgraph of the directed acyclic graph associated with the SLP. The random access problem can be reduced to *weighted ancestor queries* (see Section 5) on every heavy tree whose edges are weighted by the lengths of the variables that branch off from the heavy tree. Using a “biased” weighted ancestor data structure one can descend in the derivation tree in $\mathcal{O}(\log N)$ time, spending amortized constant time on each heavy tree [3]. Our main contribution is a solution of the weighted ancestor problem in the form of an SLP: Given a tree T of size n where the edges are labeled by weighted symbols, we construct a contracting SLP of size $\mathcal{O}(n)$ defining all prefixes in T , i.e. labels of paths from the root to some node. The special case of defining all prefixes of a weighted string by a weight-balanced SLP of linear size (i.e. T is a path) was solved in [10]; however, the constructed SLP only satisfies a weaker balancedness condition.

To solve finger search efficiently, Bille, Christiansen, Cording, and Gørtz first consider the *fringe access problem* [2]: Given a variable A and a position $1 \leq i \leq |A|$, access symbol $A[i]$, ideally in time $\mathcal{O}(\log d)$ where $d = \min\{i, |A| - i + 1\}$. For this purpose the SLP is partitioned into *leftmost* and *rightmost trees*, which produce strings of length N , $N^{1/2}$, $N^{1/4}$, $N^{1/8}$, etc. The leftmost/rightmost trees can be traversed in $\mathcal{O}(\log \log N)$ time using a $\mathcal{O}(\log \log N)$ time weighted ancestor data structure by Farach-Colton and Muthukrishnan [6]. Applying this approach to contracting SLPs one can solve fringe access in time $\mathcal{O}(\log d + \log \log |A|)$ since the trees have $\mathcal{O}(\log N)$ height, for which one can answer weighted ancestor queries in constant time using a predecessor data structure by Pătraşcu-Thorup [20]. Using additional weighted ancestor structures, we can reduce the term $\log \log |A|$ to $\log^{(t)} N$.

2 Straight-line programs

A *context-free grammar* $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ consists of a finite set \mathcal{V} of *variables*, an alphabet Σ of *terminal symbols*, where $\mathcal{V} \cap \Sigma = \emptyset$, a finite set \mathcal{R} of *rules* $A \rightarrow u$ where $A \in \mathcal{V}$ and $u \in (\mathcal{V} \cup \Sigma)^*$ is a *right-hand side*, and a *start variable* $S \in \mathcal{V}$. The set of *symbols* is $\mathcal{V} \cup \Sigma$. We call \mathcal{G} a *straight-line program (SLP)* if every variable occurs exactly once on the left-hand side of a rule and there exists a linear order $<$ on \mathcal{V} such that $A < B$ whenever B occurs on the right-hand side of a rule $A \rightarrow u$. This ensures that every variable A derives a unique string $\llbracket A \rrbracket \in \Sigma^*$. We also write $|A|$ for $|\llbracket A \rrbracket|$. A string $s \in \Sigma^*$ is *defined* by \mathcal{G} if $\llbracket A \rrbracket = s$ for some $A \in \mathcal{V}$. The *size* of \mathcal{G} is the total length of all right-hand sides in \mathcal{G} . We denote by $\text{height}(A)$ the height of the derivation tree rooted in A . The height of \mathcal{G} is $\text{height}(S)$. We define the directed acyclic graph $\text{dag}(\mathcal{G}) = (\mathcal{V} \cup \Sigma, E)$ where E is a multiset of edges, containing for every rule $A \rightarrow \beta_1 \dots \beta_k$ in \mathcal{R} the edges $(A, \beta_1), \dots, (A, \beta_k)$. An SLP \mathcal{G} can be transformed in linear time into an SLP \mathcal{G}' in *Chomsky normal form* which defines all strings that \mathcal{G} defines, i.e. each rule is of the form $A \rightarrow BC$ or $A \rightarrow a$ where $A, B, C \in \mathcal{V}$ and $a \in \Sigma$.

An SLP is α -*balanced*, for some constant $0 < \alpha \leq 1/2$, if it is in Chomsky normal form and for all rules $A \rightarrow BC$ both $|B|/|A|$ and $|C|/|A|$ lie between α and $1 - \alpha$. An *AVL-grammar* is an SLP in Chomsky normal form where for all rules $A \rightarrow BC$ we have $|\text{height}(B) - \text{height}(C)| \leq 1$. An SLP in Chomsky normal form is (α, β) -*path balanced*, for some constants $0 < \alpha \leq \beta$, if for every variable A the length of every root-to-leaf path in the derivation tree is between $\alpha \log |A|$ and $\beta \log |A|$. Observe that every α -balanced SLP is $(1/\log(\alpha^{-1}), 1/\log((1 - \alpha)^{-1}))$ -path balanced and AVL-grammars are $(0.5, 2)$ -path balanced. The latter follows from the fact that the height decreases at most by 2 when going from an

AVL-tree to an immediate subtree. There are algorithms that compute for given a string w an α -balanced SLP [5] and an AVL-grammar [22] of size $\mathcal{O}(g \log N)$ where g is the size of a smallest SLP for w . We show that these bounds are optimal even for path balanced SLPs: There are strings for which the smallest path balanced SLPs have size $\Omega(g \log N)$.

► **Theorem 4.** *There exists a family of strings $(s_n)_{n \geq 1}$ over $\{a, b\}$ such that $|s_n| = \Omega(2^n)$, s_n has an SLP of size $\mathcal{O}(n)$ and every (α, β) -path balanced SLP has size $\Omega(n^2)$.*

Proof. First we use an unbounded alphabet. Let $s_n = b_1 a^{2^n} b_2 a^{2^n} \dots b_{n-1} a^{2^n} b_n$, which has an SLP of size $\mathcal{O}(n)$ with the rules $S \rightarrow b_1 A_n b_2 A_n \dots b_n$, $A_0 \rightarrow a$ and $A_i \rightarrow A_{i-1} A_{i-1}$ for all $1 \leq i \leq n$. Consider an (α, β) -path balanced SLP \mathcal{G} for s_n . We will show that $\text{dag}(\mathcal{G})$ has $\Omega(n^2)$ edges. Let $1 \leq i \leq n$ and consider the unique path in $\text{dag}(\mathcal{G})$ from the starting variable to b_i . Let π_i be the suffix path starting in the lowest node A_i such that $\llbracket A_i \rrbracket$ contains some symbol b_j with $i \neq j$. Therefore $|A_i| \geq 2^n$. Since \mathcal{G} is (α, β) -path balanced π_i has length $\geq \alpha n$. Since all paths π_i are edge-disjoint it follows that \mathcal{G} has size $\Omega(n^2)$.

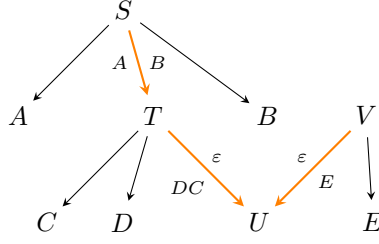
For a binary alphabet define the separator $T_i = b a^{2^{i-2}} b a^{2^{i-1}} b$ for $1 \leq i \leq n$ and define $s_n = T_1 a^{2^n} T_2 \dots T_{n-1} a^{2^n} T_n$ of length $\Omega(2^n)$. The string s_n also has an SLP of size $\mathcal{O}(n)$. Consider an (α, β) -path balanced SLP \mathcal{G} for s_n . Let $1 \leq i \leq n$ and consider the unique path ρ_i in $\text{dag}(\mathcal{G})$ from the starting variable to the symbol b in the middle of the separator $T_i = b a^{2^{i-2}} b a^{2^{i-1}} b$. Let B_i be the lowest node on ρ_i such that $\llbracket B_i \rrbracket$ contains either $b a^{2^{i-2}} b$ or $b a^{2^{i-1}} b$. Since the successor of B_i on ρ_i produces a string strictly shorter than $|T_i| \leq 4n$, the suffix path of ρ_i starting in B_i has length at most $1 + \beta \log(4n) = \mathcal{O}(\log n)$. Let A_i be the lowest ancestor of B_i on ρ_i such that $\llbracket A_i \rrbracket$ contains a symbol from a separator T_j for $i \neq j$. Therefore $|A_i| \geq 2^n$ and hence the suffix path of ρ_i starting in A_i has length at least $\alpha \log(2^n) = \alpha n = \Omega(n)$. Thus, the path π_i from A_i to B_i has length $\Omega(n) - \mathcal{O}(\log n) = \Omega(n)$. All paths π_i are edge-disjoint since for any edge (X, Y) in π_i , $\llbracket Y \rrbracket$ is of the form $a^\ell b a^{2^{i-2}} b a^r$ or $a^\ell b a^{2^{i-1}} b a^r$. This implies that \mathcal{G} has size $\Omega(n^2)$. ◀

We define contracting SLPs over a weighted alphabet, i.e. an alphabet Γ equipped with a function $\|\cdot\|: \Gamma \rightarrow \mathbb{N} \setminus \{0\}$, which is extended additively to Γ^* . The standard weight function is the length function $|\cdot|$. A symbol β occurring in a weighted string s is *heavy in s* if $\|\beta\| > \|s\|/2$; otherwise it is *light in s* . Consider an SLP $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ over a weighted alphabet Σ . We define $\|A\| = \|\llbracket A \rrbracket\|$ for $A \in \mathcal{V}$. A symbol $\beta \in \mathcal{V} \cup \Sigma$ is a *heavy child* of $A \in \mathcal{V}$ if β is heavy on the right-hand side of the rule $A \rightarrow u$. We also call β a *heavy symbol*. A rule $A \rightarrow u$ is *contracting* if u contains no heavy variables, i.e. every variable B occurring in u satisfies $\|B\| \leq \|A\|/2$. Let us emphasize that heavy *terminal* symbols from Σ are permitted in contracting rules. If all rules in \mathcal{G} are contracting we call \mathcal{G} contracting. If B occurs heavily in a rule $A \rightarrow u B v$ and the rule $B \rightarrow x$ is contracting we can *expand* the occurrence of B and obtain a contracting rule $A \rightarrow u x v$.

3 Transformation into contracting SLPs

A *labeled tree* $T = (V, E, \gamma)$ is a rooted tree where each edge $e \in E$ is labeled by a string $\gamma(e)$ over a weighted alphabet Γ . A *prefix* in T is the labeling of a path starting from the root. The first step towards proving Theorem 1 is a reduction to the following problem: Given a labeled tree T , construct a contracting SLP over Γ of size $\mathcal{O}(|T|)$, defining all prefixes in T .

Decomposition into heavy trees. Consider an SLP $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$. If a rule $A \rightarrow \beta_1 \dots \beta_k$ contains a unique heavy symbol β_i then $\beta_1 \dots \beta_{i-1}$ is the *light prefix* of A and $\beta_{i+1} \dots \beta_k$ is the *light suffix* of A . The *heavy forest* $H = (\mathcal{V} \cup \Sigma, E_H)$ contains all edges (A, β) where



■ **Figure 1** An excerpt from the dag representation of an SLP. The variables S, T, U, V form a heavy tree with root U . The value of S can be split into the prefix ACD , the root U of the heavy tree, and the suffix B . Observe that the left labeling of the path from U to S is DCA , which is the reverse of the prefix ACD .

$\beta \in \mathcal{V} \cup \Sigma$ is a heavy child of $A \in \mathcal{V}$, which is a subgraph of $\text{dag}(\mathcal{G})$. Notice that the edges in H point towards the roots, i.e. if $(\alpha, \beta) \in E$ then α is a child of β in H . We define two labeling functions: The *left label* $\lambda(e)$ of an edge $e = (A, \beta)$ is the *reversed* light prefix of A and the *right label* $\rho(e)$ of e is the light suffix of A . The connected components of (H, λ) and (H, ρ) are called the *left labeled* and *right labeled heavy trees*, which can be computed in linear time from \mathcal{G} . If β is the root of a heavy tree containing a variable A we can factorize $\llbracket A \rrbracket$ into the reversed left labeling from A to its root β in H , the value of β , and the right labeling of the path from β to A . In that way one can redefine every variable using SLPs which define all prefixes in the left labeled and the right labeled heavy trees.

► **Proposition 5.** *Given an SLP \mathcal{G} and contracting SLPs \mathcal{H}_L and \mathcal{H}_R defining all prefixes of all left labeled and right labeled heavy trees of \mathcal{G} . Let g be the total number of variables in the SLPs and r be the maximal length of a right-hand side. One can compute in linear time a contracting SLP \mathcal{G}' which defines all strings that \mathcal{G} defines, has $\mathcal{O}(g)$ variables and right-hand sides of length $\mathcal{O}(r)$.*

The goal of this section is to prove the following result.

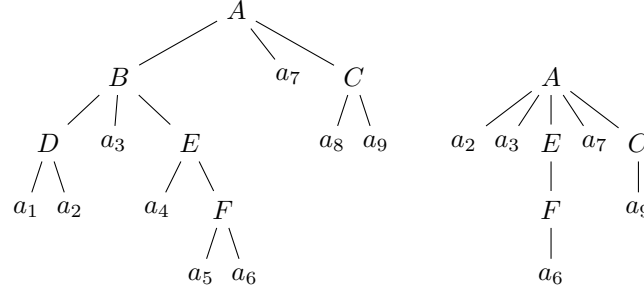
► **Theorem 6.** *Given a labeled tree T with n edges and labels of length $\leq \ell$, one can compute in linear time a contracting SLP with $\mathcal{O}(n)$ variables and right-hand sides of length $\mathcal{O}(\ell)$ defining all prefixes in T .*

Together with Proposition 5 it implies Theorem 1. We can always assume that every edge in T is labeled by a single symbol: Edges labeled by ε can clearly be contracted. Edge labels u of length > 1 are replaced by a new symbol X_u of weight $\|u\|$, which can be replaced by u again in the constructed SLP. We will also assume that all symbols in T are distinct.

Prefixes of weighted strings. We start with the case where the tree is a path, i.e. we need to define all prefixes of a weighted string s using $\mathcal{O}(|s|)$ contracting rules. The following theorem refines [10, Lemma III.1] where only the path length from a prefix variable S_i to a symbol a_j in the derivation tree was bounded by $\mathcal{O}(1 + \log \frac{\|S_i\|}{\|a_j\|})$.

► **Theorem 7.** *Given a weighted string s of length n one can compute in linear time a contracting SLP with $\mathcal{O}(n)$ variables with right-hand sides of length at most 10 that defines all nonempty prefixes of s .*

Let us illustrate the difficulty of defining all prefixes with contracting rules. Consider the weighted string $s = a_1 \dots a_n$ where symbol a_i has weight 2^{n-i} . Since in every factor $a_i \dots a_j$ the left-most symbol a_i is heavy, every rule for $a_i \dots a_j$ must split off the first symbol. If for



■ **Figure 2** The derivation tree D of a base SLP and the modified tree D_0 containing all symbols which are not a left-most child in D .

every prefix we would only repeatedly split off the first symbol we would create $\Omega(n^2)$ many variables. This shows that there is no better solution with right-hand sides of length ≤ 2 . However, using longer rules we can simultaneously reduce both the weight (in a contracting fashion) and the length.

First we recursively construct a contracting “base” SLP $\mathcal{B} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ for the weighted string $s = a_1 \dots a_n$. It will have the additional property of being *left-heavy*, i.e. for every rule $A \rightarrow \beta_1 \dots \beta_k$ and all $2 \leq i \leq k$ with $\beta_i \in \mathcal{V}$ we have $\|\beta_1 \dots \beta_{i-1}\| \geq \|\beta_i\|$. Let us emphasize that the condition does not apply when β_i is a terminal symbol. The case $n = 1$ is clear. If $n > 1$ we factorize $s = ua_iv$ such that $u, v \in \Sigma^*$ have weight at most $\|s\|/2$. Next factorize $v = v_1v_2$ such that $|v_1|$ and $|v_2|$ differ at most by one. We add the rule $S \rightarrow Ua_iV_1V_2$ to the SLP, possibly omitting variables if some of the strings u, v_1, v_2 are empty. Finally, we recursively define the variables U, V_1 and V_2 . The SLP \mathcal{B} is clearly contracting, has at most n variables, since every variable can be identified with the unique symbol $a \in \Sigma$ on its right-hand side, and its right-hand sides have length at most 4. Notice that the rule $S \rightarrow Ua_iV_1V_2$ is left-heavy since $\|ua_i\| > \|s\|/2 \geq \|v_1\| + \|v_2\|$.

► **Lemma 8.** *The base SLP \mathcal{B} can be computed in linear time from s .*

Consider the derivation tree D of \mathcal{B} whose node set is $\mathcal{S} = \mathcal{V} \cup \Sigma$. Let \preceq_D and \prec_D be the ancestor and the proper ancestor relation on \mathcal{S} . For all $\alpha \preceq_D \beta$ we define $\text{left}(\alpha, \beta) = u$ where $\alpha \Rightarrow_{\mathcal{B}}^* u\beta v$ is the unique derivation with $u, v \in \Sigma^*$. In the derivation tree $\text{left}(\alpha, \beta)$ is the string that branches off to the left on the path from α to β . Notice that every proper nonempty prefix of s can be written as $\text{left}(S, a_i) = a_1 \dots a_{i-1}$. For $\alpha \in \mathcal{S} \setminus \{S\}$ occurring in the unique rule $\alpha' \rightarrow u\alpha v$ we define the *left sibling string* $\text{lsib}(\alpha) = u$. It satisfies $\text{lsib}(\alpha) \Rightarrow_{\mathcal{B}}^* \text{left}(\alpha', \alpha)$. Notice that we can have $\text{left}(\alpha, \beta) = \text{left}(\alpha', \beta')$ for different pairs $(\alpha, \beta), (\alpha', \beta')$. For a unique description we define the set of nodes $\mathcal{S}_0 \subseteq \mathcal{S}$ which are not a left-most child in D , i.e. symbols α such that $\alpha = S$ or $\text{lsib}(\alpha) \neq \varepsilon$. In particular S belongs to \mathcal{S}_0 . Observe that $\text{left}(\alpha, \beta) = \text{left}(\alpha', \beta')$ where α' and β' are the lowest ancestors of α and β , respectively, that belong to \mathcal{S}_0 . In particular, every proper nonempty prefix of s is of the form $\text{left}(\alpha, \beta)$ for some $\alpha, \beta \in \mathcal{S}_0$. Let D_0 be the unique unordered tree with node set \mathcal{S}_0 whose ancestor relation is the ancestor relation of D restricted to \mathcal{S}_0 . Figure 2 shows an example of a tree D with the modified tree D_0 .

We will introduce variables $L_{\alpha, \beta}$ for the strings $\text{left}(\alpha, \beta)$. The variable $L_{\alpha, \beta}$ can be defined using $L_{\alpha', \beta'}$ where α' is a child of α in D_0 and β' is the parent of β in D_0 . To achieve the $\mathcal{O}(n)$ bound we will restrict to variables $L_{\alpha, \beta}$ that are used in the derivation of a prefix variable, namely $\mathcal{L} = \{L_{\alpha, \beta} \mid \alpha, \beta \in \mathcal{S}_0, \alpha \prec \beta, \text{level}(\alpha) \leq \text{height}(\beta)\}$. Here $\text{level}(\alpha)$ refers to distance from α to the root in D_0 , and $\text{height}(\beta)$ is the height of the subtree of D_0 below β .

► **Lemma 9.** *We can compute in linear time a contracting SLP $\mathcal{G} = (\mathcal{V} \cup \mathcal{L}, \Sigma, \mathcal{R} \cup \mathcal{Q}, S)$ with right-hand sides of constant length such that $\llbracket L_{\alpha, \beta} \rrbracket = \text{left}(\alpha, \beta)$ for all $L_{\alpha, \beta} \in \mathcal{L}$.*

We have seen that \mathcal{G} defines all nonempty prefixes (S derives s and every proper nonempty prefix is defined by some variable L_{S, a_i}). To finish the proof of Theorem 7 remains to show that \mathcal{G} has $\mathcal{O}(n)$ variables. The SLP \mathcal{G} consists of n variables from the base SLP \mathcal{B} and the variables in \mathcal{L} . A variable $L_{\alpha, \beta} \in \mathcal{L}$ is determined by β and the level of α , which is an integer between 0 and the height of β in D_0 . Hence it suffices to show that $\sum_{\beta \in S_0} \text{height}(\beta)$ is $\mathcal{O}(n)$. This follows from the fact that every node in D_0 has logarithmic height in its leaf size.

Prefixes in trees. For Theorem 6 we will construct an SLP \mathcal{G} for the prefixes in T , which will not be contracting in general. Still, its heavy forest is a disjoint union of *caterpillar trees*, i.e. trees where every node has at most one child which is not a leaf. Put differently, every heavy tree of \mathcal{G} consists of a central path $\alpha_1, \dots, \alpha_m$ such that every α_i occurs at most once heavily in a rule $A \rightarrow u$ where A is heavy, namely $A = \alpha_{i-1}$. We first extend Theorem 7 to caterpillar trees and then apply Proposition 5 to \mathcal{G} , concluding the proof of Theorem 6.

► **Proposition 10.** *Given a labeled caterpillar tree T with n edges and labels of length $\leq \ell$, one can compute a contracting SLP \mathcal{G} defining all nonempty prefixes in T such that \mathcal{G} has $\mathcal{O}(n)$ variables and right-hand sides of length $\mathcal{O}(\ell)$.*

► **Proposition 11.** *Given a labeled tree T with n edges we can compute an SLP \mathcal{G} defining all nonempty prefixes in T such that*

- (a) \mathcal{G} has $4n$ variables and right-hand sides of length ≤ 6 ,
- (b) the subgraph of $\text{dag}(\mathcal{G})$ induced by the set of heavy symbols is a disjoint union of paths.

Proof sketch. We proceed by induction on n . Let us assume a tree $T = (V, E, \omega)$ with $n \geq 2$ edges. We partition E into maximal *unary paths* $\pi = (v_0, \dots, v_k)$, where $k \geq 1$, and v_1, \dots, v_{k-1} have degree one. For every such a path π we create an SLP \mathcal{G}_π with the rules $P_{v_0, v_1} \rightarrow \omega(v_0, v_1)$ and $P_{v_0, v_i} \rightarrow P_{v_0, v_{i-1}} \omega(v_{i-1}, v_i)$ for $2 \leq i \leq k$. We contract every such a maximal unary path into a single edge (v_0, v_k) labeled by the variable P_{v_0, v_k} and remove all leaves. This new tree T' has at most $n/2$ many edges. Let $V' \subseteq V$ be the node set of T' .

For $v \in V'$ let $d(v)$ be the weight of the path from the root to v in T' and define $\text{rk}(v) = \inf\{k \in \mathbb{Z} \mid d(v) \leq 2^k\}$. Let \hat{v} be the highest ancestor of v in T' with $\text{rk}(v) = \text{rk}(\hat{v})$, called the *peak node* of v . We partition T' into subtrees consisting of nodes with the same peak node, and apply the construction recursively on each part. Let \mathcal{G}' be the union of all obtained SLPs with at most $4 \cdot n/2 \leq 2n$ variables. For every node $v \in V'$ which is not a peak node \mathcal{G}' contains a variable $B_{\hat{v}, v}$ where $\llbracket B_{\hat{v}, v} \rrbracket$ is the path labeling from \hat{v} to v in T' .

Let \mathcal{G} be the union of \mathcal{G}' and all SLPs \mathcal{G}_π , which has at most $n + 2n = 3n$ variables. For every $x \in V$ which is not the root of T we add a variable A_x such that $\llbracket A_x \rrbracket$ is the labeling of the path from the root to x in T . This yields $4n$ variables, as claimed. Let v be the lowest ancestor of x in T contained in V' . If v is the root then we add the rule

$$A_x \rightarrow P_{v, x}. \quad (1)$$

Now assume that v is not the root and hence \hat{v} is also not the root, since the children of the root are peak nodes. Let u be the parent of \hat{v} in T' . If u is the root of T' we add the rule

$$A_x \rightarrow P_{u, \hat{v}} B_{\hat{v}, v} P_{v, x}. \quad (2)$$

Otherwise, u and \hat{u} are not the root. Let s be the parent node of \hat{u} in T' and add the rule

$$A_x \rightarrow A_s P_{s,\hat{u}} B_{\hat{u},u} P_{u,\hat{v}} B_{\hat{v},v} P_{v,x}. \quad (3)$$

One can prove correctness by induction on $\text{rk}(v)$. It remains to prove property (b) of the statement. One can observe that the A - and B -variables are light in the rules (2) and (3). Consider a maximal unary path $\pi = (v_0, \dots, v_k)$. The variable P_{v_0, v_i} for $1 \leq i \leq k-1$ only occurs in the rule of $P_{v_0, v_{i-1}}$. The variable P_{v_0, v_k} can occur on the right-hand sides of (1), (2) and (3), but the corresponding left-hand side A_x is not heavy. By induction hypothesis P_{v_0, v_k} is the heavy child of at most one heavy variable in \mathcal{G}' . This concludes the proof. \blacktriangleleft

4 Navigation in FSLP-compressed trees

As a simple application we extend the navigation data structure on FSLP-compressed trees [21] by the operation which moves to the i -th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node. This is established by applying Theorem 1 to the substructure of the FSLP that compresses forests horizontally.

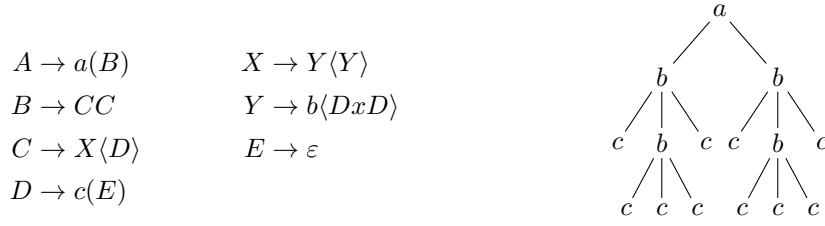
SLP navigation. The navigation data structure on FSLPs is based on a navigation data structure on (string) SLPs from [18], which extends the data structure from [12] from one-way to two-way navigation. The data structure represents a position $1 \leq i \leq |A|$ in a variable A by a data structure $\sigma(A, i)$, that we will call *pointer*, which is a compact representation of the path in the derivation tree from A to the leaf corresponding to position i .

► **Theorem 12** ([18]). *A given SLP \mathcal{S} can be preprocessed in $\mathcal{O}(|\mathcal{S}|)$ time and space so that the following operations are supported in constant time:*

- *Given a variable A , compute $\sigma(A, 1)$ or $\sigma(A, |A|)$.*
- *Given $\sigma(A, i)$, compute $\sigma(A, i-1)$ or $\sigma(A, i+1)$, or return \perp if the position is invalid.*
- *Given $\sigma(A, i)$, return the symbol at position i in A .*

Furthermore, a single pointer $\sigma(A, i)$ uses $\mathcal{O}(\text{height}(A))$ space and can be computed in time $\mathcal{O}(\text{height}(A))$ for a given pair (A, i) .

Forest straight-line programs. In this section we use the natural term representation for forests. Let Σ be an alphabet of node labels. The set of *forests* is defined inductively as follows: The concatenation of $n \geq 0$ forests is a forest (this includes the empty forest ε), and, if $a \in \Sigma$ and t is a forest, then $a(t)$ is a forest. A *context* is a forest over $\Sigma \cup \{x\}$ where x occurs exactly once and this occurrence is at a leaf node. If f is a context and g is a forest or a context then $f\langle g \rangle$ is obtained by replacing the unique occurrence of x in f by g . A *forest straight-line program (FSLP)* $\mathcal{G} = (\mathcal{V}_0, \mathcal{V}_1, \Sigma, \mathcal{R}, S)$ consists of finite sets of forest variables \mathcal{V}_0 and context variables \mathcal{V}_1 , the alphabet Σ , a finite set of rules \mathcal{R} , and a start variable $S \in \mathcal{V}_0$. The rules contain arbitrary applications of horizontal concatenation and substitutions of forest and context variables. We restrict ourselves to rules in a certain *normal form*, which can be established in linear time with a constant factor size increase [11]. The normal form assumes a partition $\mathcal{V}_0 = \mathcal{V}_0^\top \cup \mathcal{V}_0^\perp$ where \mathcal{V}_0^\top -variables produce trees whereas \mathcal{V}_0^\perp -variables



■ **Figure 3** An example FSLP with the variables $\mathcal{V}_0^\perp = \{A, C, D\}$, $\mathcal{V}_0^\top = \{B, E\}$ and $\mathcal{V}_1 = \{X, Y\}$. The tree defined by A is displayed on the right.

produce forests with arbitrarily many trees. The rules in \mathcal{R} have one of the following forms:

$A \rightarrow \varepsilon$	where $A \in \mathcal{V}_0^\top$,
$A \rightarrow BC$	where $A \in \mathcal{V}_0^\top$ and $B, C \in \mathcal{V}_0$,
$A \rightarrow a(B)$	where $A \in \mathcal{V}_0^\perp$, $a \in \Sigma$, and $B \in \mathcal{V}_0$,
$A \rightarrow X\langle B \rangle$	where $A, B \in \mathcal{V}_0^\perp$ and $X \in \mathcal{V}_1$,
$X \rightarrow Y\langle Z \rangle$	where $X, Y, Z \in \mathcal{V}_1$
$X \rightarrow a(LxR)$	where $X \in \mathcal{V}_1$, $a \in \Sigma$, and $L, R \in \mathcal{V}_0$,

Every variable $A \in \mathcal{V}_0$ derives a forest $\llbracket A \rrbracket$ and every variable $X \in \mathcal{V}_1$ derives a context $\llbracket X \rrbracket$, see [11] for formal definitions. An example FSLP for a tree is shown in Figure 3.

The normal form allows us to define two string SLPs (without start variables) that capture the horizontal and the vertical compression in \mathcal{G} . The *rib SLP* $\mathcal{G}_\square = (\mathcal{V}_0, \Sigma_\square, \mathcal{R}_\square)$ over the alphabet $\Sigma_\square = \{\underline{A} \mid A \in \mathcal{V}_0^\perp\}$ contains all rules of the form $A \rightarrow \varepsilon$ or $A \rightarrow BC$ from \mathcal{R} where $A \in \mathcal{V}_0^\top$, and the rule $A \rightarrow \underline{A}$ for all $A \in \mathcal{V}_0^\perp$. We write $\llbracket A \rrbracket_\square = \underline{A}_1 \dots \underline{A}_n$ for the string derived by A in \mathcal{G}_\square , which satisfies $\llbracket A \rrbracket = \llbracket A_1 \rrbracket \dots \llbracket A_n \rrbracket$. In the example of Figure 3 we have $\llbracket B \rrbracket_\square = \underline{C} \underline{C}$. The *spine SLP* $\mathcal{G}_\sqcap = (\mathcal{V}_0^\perp \cup \mathcal{V}_1, \Sigma_\sqcap, \mathcal{R}_\sqcap)$ is defined over the alphabet

$$\Sigma_\sqcap = \{a(B) \mid (A \rightarrow a(B)) \in \mathcal{R}\} \cup \{a(LxR) \mid (X \rightarrow a(LxR)) \in \mathcal{R}\}.$$

The set \mathcal{R}_\sqcap contains all rules $A \rightarrow a(B)$ and $X \rightarrow a(LxR)$ from \mathcal{R} . It also contains the rule $A \rightarrow X$ for all $(A \rightarrow X\langle B \rangle) \in \mathcal{R}$ where $A \in \mathcal{V}_0^\perp$, and $X \rightarrow YZ$ for all $(X \rightarrow Y\langle Z \rangle) \in \mathcal{R}$. We write $\llbracket V \rrbracket_\sqcap$ for the string derived by V in \mathcal{G}_\sqcap . If $X \in \mathcal{V}_1$ and $\llbracket X \rrbracket_\sqcap = a_1(L_1xR_1) \dots a_n(L_nxR_n)$ then $\llbracket X \rrbracket$ is the vertical composition of all contexts $a_i(\llbracket L_i \rrbracket_\sqcap x \llbracket R_i \rrbracket_\sqcap)$. In the example of Figure 3 we have $\llbracket C \rrbracket_\sqcap = b\langle Dx D \rangle b\langle Dx D \rangle$.

FSLP navigation. Now we define the data structure from [21]. It represents a node v in a tree produced by a variable $A \in \mathcal{V}_0$ by a pointer $\tau(A, v)$, which is basically a sequence of navigation pointers in the SLPs \mathcal{G}_\square and \mathcal{G}_\sqcap describing the path from the root of $\llbracket A \rrbracket$ to v . Intuitively, the pointer $\tau(A, v)$ can be described as follows. First we select the subtree of $\llbracket A \rrbracket$ which contains v , by navigating in \mathcal{G}_\square to a symbol \underline{B}_0 where $B_0 \in \mathcal{V}_0^\perp$. The tree $\llbracket B_0 \rrbracket$ is defined by a sequence of insertion rules $B_0 \rightarrow X_1\langle B_1 \rangle$, $B_1 \rightarrow X_2\langle B_2 \rangle$, \dots , $B_{k-1} \rightarrow X_k\langle B_k \rangle$, where possibly $k = 0$, and a final rule $B_k \rightarrow a(C)$. We navigate in \mathcal{G}_\sqcap in the variable B_0 from left to right. The string $\llbracket B_0 \rrbracket_\sqcap$ specifies the contexts $a_j(L_jxR_j)$ which together form the context $\llbracket X_1 \rrbracket$. If we encounter a context $a_j(L_jxR_j)$ which contains v , there are two cases. If v is the a_j -labeled root then we are done. If v is contained in either L_j or R_j then we record the direction (L or R) and continue recursively from the variable L_j or R_j . If v is not contained in the context X_1 then we reach the end of $\llbracket B_0 \rrbracket_\sqcap$, and continue searching from B_1 ,

etc. If v is contained in B_k then it is either its root or it is contained in C . In the former case, we are done; in the latter case we record the direction \mathbf{M} and continue recursively from C .

To define $\tau(A, v)$ formally, let us write $\sigma_{\sqsubseteq}(A, i)$ and $\sigma_{\sqsupset}(A, i)$ for the pointers to the i -th position of a variable $A \in \mathcal{V}_0$ in $\mathcal{G}_{\sqsubseteq}$ and \mathcal{G}_{\sqsupset} , respectively. We represent every node v in every variable $A \in \mathcal{V}_0$ by a *horizontal pointer* $\tau_{\sqsubseteq}(A, v)$. Furthermore, we represent every node v in every variable $A \in \mathcal{V}_0^{\perp}$, deriving a tree, by a *vertical pointer* $\tau_{\sqsupset}(A, v)$. The pointers are defined recursively as follows:

1. Let $A \in \mathcal{V}_0$. If $\llbracket A \rrbracket_{\sqsubseteq} = \underline{A_1} \dots \underline{A_n}$ and v is contained in $\llbracket A_i \rrbracket$ then set $\tau_{\sqsubseteq}(A, v) := \sigma_{\sqsubseteq}(A, i) \tau_{\sqsubseteq}(A_i, v)$.
2. Let $A \in \mathcal{V}_0^{\perp}$ with a rule $A \rightarrow a(B)$. If v is the root of $\llbracket A \rrbracket$ set $\tau_{\sqsupset}(A, v) := \sigma_{\sqsupset}(A, 1)$, and otherwise $\tau_{\sqsupset}(A, v) := \sigma_{\sqsupset}(A, 1) \mathbf{M} \tau_{\sqsupset}(B, v)$.
3. Let $A \in \mathcal{V}_0^{\perp}$ with a rule $A \rightarrow X \langle B \rangle$ and $\llbracket A \rrbracket_{\sqsupset} = \llbracket X \rrbracket_{\sqsupset} = a_1(L_1 x R_1) \dots a_n(L_n x R_n)$. If v is contained in $f_i = a_i(\llbracket L_i \rrbracket x \llbracket R_i \rrbracket)$ set $\tau_{\sqsupset}(A, v)$ to be $\sigma_{\sqsupset}(A, i)$, $\sigma_{\sqsupset}(A, i) \mathbf{L} \tau_{\sqsupset}(L_i, v)$ or $\sigma_{\sqsupset}(A, i) \mathbf{R} \tau_{\sqsupset}(R_i, v)$, depending whether v is the root of f_i or is contained in $\llbracket L_i \rrbracket$ or $\llbracket R_i \rrbracket$. If v is contained in $\llbracket B \rrbracket$ set $\tau_{\sqsupset}(A, v) := \sigma_{\sqsupset}(A, n) \tau_{\sqsupset}(B, v)$.

For the navigation we only use the horizontal pointers and write $\tau(A, v)$ instead of $\tau_{\sqsubseteq}(A, v)$.

► **Theorem 13** ([21]). *A given FSLP \mathcal{G} can be preprocessed in $\mathcal{O}(|\mathcal{G}|)$ time and space so that the following operations are supported in constant time:*

- *Given a variable A , compute $\tau(A, v)$ where v is the root of the first/last tree in $\llbracket A \rrbracket$.*
- *Given $\tau(A, v)$, compute $\tau(A, v')$ where v' is the parent, first/last child or left/right sibling of v , or return \perp if it does not exist.*
- *Given $\tau(A, v)$, return the symbol of node v .*

Navigation to a child. We extend Theorem 13 by the operation which, given a pointer $\tau(S, v)$ and a number $1 \leq j \leq d$, where v has degree d , moves the pointer to the j -th child of v in $\mathcal{O}(\log d)$ time. To this end we apply Theorem 1 to $\mathcal{G}_{\sqsubseteq}$ so that every variable $A \in \mathcal{V}_0$ in the rib SLP has height $\mathcal{O}(\log |\llbracket A \rrbracket_{\sqsubseteq}|)$, by adding only $\mathcal{O}(g)$ new variables. In particular, we can compute a pointer $\sigma_{\sqsubseteq}(A, i)$ in $\mathcal{O}(\log |\llbracket A \rrbracket_{\sqsubseteq}|)$ time by Theorem 12. Furthermore, we compute the length $|\llbracket A \rrbracket_{\sqsubseteq}|$ for all $A \in \mathcal{V}_0$ in linear time.

Suppose we are given a pointer $\tau(S, v)$ to a node v with degree d for some variable $S \in \mathcal{V}_0$. We show how to compute $\tau(S, v_j)$ where v_j is the j -th child of v in $\mathcal{O}(\log d)$ time.

1. In the first case the last pointer in $\tau(S, v)$ is $\sigma_{\sqsubseteq}(A, 1)$ where the rule of $A \in \mathcal{V}_0^{\perp}$ is of the form $A \rightarrow a(B)$. Here B derives the forest below the a -node and we need to move to the root of the j -th tree in the forest. We compute the pointer $\sigma_{\sqsubseteq}(B, j)$ in $\mathcal{O}(\log |\llbracket B \rrbracket_{\sqsubseteq}|) \leq \mathcal{O}(\log d)$ time. Then we query the symbol $\underline{B_j}$ at pointer $\sigma_{\sqsubseteq}(B, j)$ and compute the pointer $\sigma_{\sqsupset}(B_j, 1)$ in constant time. Then we obtain $\tau(S, v_j) = \tau(S, v) \mathbf{M} \sigma_{\sqsubseteq}(B, j) \sigma_{\sqsupset}(B_j, 1)$.
2. In the second case the last pointer in $\tau(S, v)$ is $\sigma_{\sqsupset}(A, i)$ where the rule of $A \in \mathcal{V}_0^{\perp}$ is of the form $A \rightarrow X \langle B \rangle$. We query the symbol $a_i(L_i x R_i)$ at pointer $\sigma_{\sqsupset}(A, i)$. The j -th child v_j is either in L_i , R_i or at the position of the parameter x . If $j = |\llbracket L_i \rrbracket_{\sqsubseteq}| + 1$ we replace $\sigma_{\sqsupset}(A, i)$ by $\sigma_{\sqsupset}(A, i + 1)$ in constant time. If this is not successful then v_j is the root of B and we have $\tau(S, v_j) = \tau(S, v) \sigma_{\sqsupset}(B, 1)$, which can be computed in constant time. If $j \leq |\llbracket L_i \rrbracket_{\sqsubseteq}|$ we compute $\sigma_{\sqsubseteq}(L_i, j)$ in $\mathcal{O}(\log |\llbracket L_i \rrbracket_{\sqsubseteq}|) \leq \mathcal{O}(\log d)$ time. We query the symbol $\underline{B_j}$ at $\sigma_{\sqsubseteq}(L_i, j)$ and compute $\sigma_{\sqsupset}(B_j, 1)$ in constant time. Then we have $\tau(S, v_j) = \tau(S, v) \mathbf{L} \sigma_{\sqsubseteq}(L_i, j) \sigma_{\sqsupset}(B_j, 1)$. If $j \geq |\llbracket L_i \rrbracket_{\sqsubseteq}| + 2$ we proceed similarly using $\sigma_{\sqsubseteq}(R_i, j - |\llbracket L_i \rrbracket_{\sqsubseteq}| - 1)$.

Remarks. In its original form the SLP navigation data structure from [18] is non-persistent, i.e. the operations modify the given pointer. However, it is not hard to adapt the structure so that an operation returns a fresh pointer, by representing paths in the derivation tree using

linked lists that share common prefixes. In a similar fashion, Theorem 2 can be adapted so that a pointer is not modified by a navigation step.

Finally, let us comment on the space consumption of a single pointer in Theorem 2. A single pointer $\tau(A, v)$ consists of a sequence of pointers in \mathcal{G}_{\square} and \mathcal{G}_{\sqcap} that almost describes a path in the derivation tree of A in \mathcal{G} . The sequence may contain pointers $\sigma_{\square}(A, n)$ that point to the lowest node above the parameter of a context $\llbracket X \rrbracket$. However, in the representation of [18] such a pointer $\sigma_{\square}(A, n)$ only uses $\mathcal{O}(1)$ space, since it is a rightmost path in the derivation tree of A in \mathcal{G}_{\square} . Therefore $\tau(A, v)$ uses $\mathcal{O}(\text{height}(A))$ space where $\text{height}(A)$ is the height of the derivation tree of A in \mathcal{G} . By [10, Theorem VII.3] we can indeed assume that the FSLP \mathcal{G} has $\mathcal{O}(\log N)$ height while retaining the size bound of $\mathcal{O}(|\mathcal{G}|)$. We also need the fact that the transformation into the normal form increases the height only by a constant factor. However, since the application of Theorem 1 to the rib SLP may possibly increase the total height of the FSLP \mathcal{G} by more than a constant factor, it is unclear whether Theorem 2 can be achieved with $\mathcal{O}(\log N)$ sized pointers.

5 Finger search in SLP-compressed strings

In this section we present our solution (Theorem 3) for the finger search problem using contracting SLPs. Our finger data structure is an *accelerated path*, which compactly represents the path from root to the finger in the derivation tree using precomputed forests on the dag of the SLP. To move the finger we ascend to some variable on the path, branch off from the path, and descend in a subtree while computing the new accelerated path. We can maintain the accelerated path in a dynamic predecessor structure with constant update and query time, thanks to the $\mathcal{O}(\log N)$ height of the SLP. We follow the approach of [2] and present an improved $\mathcal{O}(tg)$ space solution for the *fringe access problem*: Given a variable A and a position $1 \leq i \leq |A|$, we can access the i -th symbol of $\llbracket A \rrbracket$ in time $\mathcal{O}(\log d + \log^{(t)} N)$ where $d = \min\{i, |A| - i + 1\}$ is the distance from the fringe of A , and t is any parameter.

Data structures. Recall that we assume the word RAM model with word size $w \geq \log N$ where N is the string length. Since all occurring sets and trees have size $n \leq N$ we have $w \geq \log n$ in the following. We use a dynamic predecessor data structure by Pătraşcu-Thorup, which represents a dynamic set S of $n = w^{\mathcal{O}(1)}$ many w -bit integers in space $\mathcal{O}(n)$, supporting the following updates and queries in constant time [20]: $\text{insert}(S, x) = S \cup \{x\}$, $\text{delete}(S, x) = S \setminus \{x\}$, $\text{pred}(S, x) = \max\{y \in S \mid y < x\}$, $\text{succ}(S, x) = \min\{y \in S \mid y > x\}$, $\text{rank}(S, x) = |\{y \in S \mid y < x\}|$, and $\text{select}(S, i) = x$ with $\text{rank}(S, x) = i$, if any. By enlarging the word size to $2w$ we can identify a number $x \cdot 2^w + y$, where x, y are w -bit numbers, with the key-value pair (x, y) , allowing us to store key-value pairs in the data structure sorted by their keys. We remark that all standard operations on a $2w$ -bit word RAM can be simulated by a constant number of w -bit operations. This dynamic predecessor structure is used to maintain the accelerated path to the finger. We extend the data structure by the operation $\text{split}(S, x) = \{y \in S \mid y \leq x\}$ for $\mathcal{O}(w)$ -sized sets.

► **Theorem 14** ([20]). *There is a data structure representing a dynamic set S of at most $n = \mathcal{O}(w)$ many w -bit numbers in space $\mathcal{O}(n)$ supporting the operations $\text{insert}(S, x)$, $\text{split}(S, x)$ and $\text{pred}(S, x)$ in constant time.*

A *weighted tree* T is a rooted tree where each node v carries a nonnegative integer $d(v)$, called the *weighted depth*, satisfying $d(u) \leq d(v)$ for all nodes v with parent u . Given a node v and a number $p \in \mathbb{N}$, the *weighted ancestor query* (v, p) asks to return the highest ancestor

u of v with $d(u) > p$. Given a node v and $p \in \mathbb{N}$, we can also compute the highest ancestor u of v where the *weighted distance* $d(u, v) = d(v) - d(u)$ is less than p , by the weighted ancestor query $(v, d(v) - p)$. In our application the edges have nonnegative weights and the weighted depth of a node is computed as the sum of all edge weights on the path from the root.

Kopelowitz and Lewenstein [15] showed that weighted ancestor queries on a tree of size n can be answered in time $\mathcal{O}(\text{pred}(n) + \log^* n)$ where $\text{pred}(n)$ is the query time of a predecessor data structure. It was claimed in [14] that the $\log^* n$ -term can be eliminated without giving an explicit proof. We refer to [9, Proposition 17] for a proof in the setting where $n \leq w$ using the predecessor structure from [20]. Furthermore, we can also support constant time weighted ancestor queries if the tree height is $\mathcal{O}(w)$.

► **Proposition 15.** *A weighted tree T with n nodes and height $h = \mathcal{O}(w)$ can be preprocessed in $\mathcal{O}(n)$ space and time so that weighted ancestor queries can be answered in constant time.*

The fringe access problem. Consider an SLP \mathcal{G} with the variable set \mathcal{V} containing g variables for a string of length N . Using Theorem 1 we assume that \mathcal{G} is in Chomsky normal form and that every variable A has height $\mathcal{O}(\log |A|)$. We precompute in linear time the length of all variables in \mathcal{G} . To simplify notation we assume that the variables B and C in all rules $A \rightarrow BC$ are distinct, which can be established by doubling the number of variables. We assign to each edge e in $\text{dag}(\mathcal{G})$ a *left weight* $\lambda(e)$ and a *right weight* $\rho(e)$: For every rule $A \rightarrow BC$ in \mathcal{G} , the edge $e = (A, B)$ has left weight $\lambda(e) = 0$ and right weight $\rho(e) = |C|$, whereas the edge $e = (A, C)$ has left weight $\rho(e) = |B|$ and right weight $\lambda(e) = 0$.

Let \mathcal{F} be a finite set of subforests of $\text{dag}(\mathcal{G})$ with node set \mathcal{V} whose edges point towards the roots (as for example in the heavy forest). The forests will be computed later in Proposition 16. For every forest $F \in \mathcal{F}$ we define two edge-weighted versions F_L and F_R where the edges inherit the left weights and the right weights from $\text{dag}(\mathcal{G})$, respectively, yielding $2|\mathcal{F}|$ many weighted forests. Let $\lambda_F(A)$ and $\rho_F(A)$ be the weighted depths of A in F_L and F_R , respectively. In $\mathcal{O}(|\mathcal{F}| \cdot g)$ time we compute for all $A \in \mathcal{V}_0$ the weighted depths $\lambda_F(A)$ and $\rho_F(A)$ and the root $\text{root}_F(A)$ of the subtree of F containing A . We write $\lambda_F(A, B)$ and $\rho_F(A, B)$ for the weighted distances between A and B in F_L and F_R , respectively. We preprocess all $2|\mathcal{F}|$ weighted forests in time and space $\mathcal{O}(|\mathcal{F}| \cdot g)$ to support weighted ancestor queries in constant time according to Proposition 15. This is possible because the height of the forests is $\mathcal{O}(\log N) = \mathcal{O}(w)$.

We denote by $\langle A, i \rangle$ the state in which we aim to compute a compact representation of the path from A to the i -th leaf in the derivation tree of A . Starting from state $\langle A, i \rangle$ we can take short steps and long steps. A *short step* considers the rule of A : If it is a terminal rule $A \rightarrow a$ we have found the symbol a . If it is a binary rule $A \rightarrow BC$ we compare i with $|B|$: If $i \leq |B|$ then the short step leads to $\langle B, i \rangle$, and otherwise to $\langle C, i - |B| \rangle$. A *left long step in* $F \in \mathcal{F}$ is possible if $i \leq \lambda_F(A) + |\text{root}_F(A)|$. Put differently, the path from A to $A[i]$ branches off to the left on the path from A to $\text{root}_F(A)$, or continues below $\text{root}_F(A)$. We determine the highest ancestor X of A in F_L with $\lambda_F(A, X) < i$ and move to $\langle X, i - \lambda_F(A, X) \rangle$. Using the weighted ancestor data structure on F the variable X can be determined in constant time. Symmetrically, a *right long step in* F is possible if $|A| - i + 1 \leq \rho_F(A) + |\text{root}_F(A)|$. Put differently, the path from A to $A[i]$ branches off to the right on the path from A to $\text{root}_F(A)$, or continues below $\text{root}_F(A)$. After finding the highest ancestor X of A in F_R with $\rho_F(A, X) < |A| - i + 1$ we move to $\langle X, |A| - i + 1 - \rho_F(A, X) \rangle$.

If we take a long step in a forest F then a subsequent short step moves us from one subtree in F to a different subtree, by maximality of the answer from the weighted ancestor query. A sequence of short and long steps is summarized in an *accelerated path* (e_1, \dots, e_m) of

short and long edges. A *short edge* is an edge (A, B) in $\text{dag}(\mathcal{G})$ whereas a *long edge* is a triple (A, F, B) such that $F \in \mathcal{F}$ contains a (unique) path from A to B . In the triple (A, F, B) we store only an identifier of F instead of the forest itself. The left weight and the right weight of a long edge $e = (A, F, B)$ are $\lambda(e) = \lambda_F(A, B)$ and $\rho(e) = \rho_F(A, B)$, respectively.

► **Proposition 16.** *Let $t \geq 1$. One can compute and preprocess in $\mathcal{O}(tg)$ time a set of forests \mathcal{F} with $|\mathcal{F}| = \mathcal{O}(t)$ so that given a variable A and a position $1 \leq i \leq |A|$, one can compute an accelerated path from A to $A[i]$ in time $\mathcal{O}(\log d + \log^{(t+1)} N)$ where $d = \min\{i, |A| - i + 1\}$.*

Proof sketch. First let us assume that $i \leq |A|/2$. To this end we construct forests $\mathcal{F} = \{F_0, \dots, F_{t-1}\}$ in $\mathcal{O}(tg)$ time so that an accelerated path from A to $A[i]$ can be computed in time $\mathcal{O}(\log i + \log^{(t+1)} N)$. For all $F \in \mathcal{F}$ we construct two constant time weighted ancestor data structures (for F_L and F_R), and compute $\lambda_F(A)$, $\rho_F(A)$ and $\text{root}_F(A)$ for all $A \in \mathcal{V}$.

The simple algorithm which only uses short steps takes time $\mathcal{O}(\log |A|)$. We first improve the running time to $\mathcal{O}(\log i + \log \log |A|)$. Let $\text{rk}(A) = \min\{k \in \mathbb{N} : |A| \leq 2^{2^k}\}$, which is at most $1 + \log \log |A|$. The forest F_0 contains for every rule $A \rightarrow BC$ in \mathcal{G} either the edge (A, B) , if $\text{rk}(A) = \text{rk}(B)$, or the edge (A, C) if $\text{rk}(A) = \text{rk}(C) > \text{rk}(B)$. If $\text{rk}(A)$ is strictly greater than both $\text{rk}(B)$ and $\text{rk}(C)$ then no edge is added for the rule $A \rightarrow BC$. To query $A[i]$ where $\text{rk}(A) = k$ we make a case distinction. If $i \leq \lambda_{F_0}(A) + |\text{root}_{F_0}(A)|$ we take a left long step in F_0 to some state $\langle X, j \rangle$ with $\text{rk}(X) < k$ and $j \leq i$, and repeat the procedure from there. Otherwise $i > |\text{root}_{F_0}(A)| > 2^{2^{k-1}} \geq \sqrt{|A|}$ and we query $A[i]$ using short steps in time $\mathcal{O}(\log |A|) \leq \mathcal{O}(\log i)$. Since the rank is reduced in the former case this procedure takes time $\mathcal{O}(\log i + k) \leq \mathcal{O}(\log i + \log \log |A|)$.

We can replace $\log \log |A|$ by $\log^{(t+1)} N$ by adding forests F_1, \dots, F_{t-1} to \mathcal{F} : The forest F_k where $1 \leq k \leq t-1$ contains for every rule $A \rightarrow BC$ in \mathcal{G} either the edge (A, B) , if $|B| > \log^{(k)} N$, or the edge (A, C) , if $|B| \leq \log^{(k)} N$ and $|C| > \log^{(k)} N$. To query $A[i]$ we compute the maximal $k \in [0, t-1]$ such that $i \leq \log^{(k)} N$. We will compute the accelerated path in time $\mathcal{O}(\log i + \log^{(k+2)} N) \leq \mathcal{O}(\log i + \log^{(t+1)} N)$. If $|A| \leq \log^{(k)} N$ we can query $A[i]$ in time $\mathcal{O}(\log i + \log \log |A|) \leq \mathcal{O}(\log i + \log^{(k+2)} N)$. If $i \leq \log^{(k)} N < |\text{root}_{F_k}(A)|$ we can take a left long step in F_k and then a short step to some state $\langle X, j \rangle$ where $|X| \leq \log^{(k)} N$ and $j \leq i$. We can query $X[j]$ in time $\mathcal{O}(\log j + \log \log |X|) \leq \mathcal{O}(\log i + \log^{(k+2)} N)$.

Finally, for every forest $F \in \mathcal{F}$ we include a mirrored right-skewed version of F , which then supports access to symbol $A[i]$ in time $\mathcal{O}(\log(|A| - i + 1) + \log^{(t+1)} N)$. ◀

Solving the finger search problem. We are ready to prove Theorem 3. We maintain an accelerated path $\pi = (e_1, \dots, e_m)$ from the start variable S to the current finger position f with its left weights and right weights as follows. Let $\ell_j = \sum_{k=1}^j \lambda(e_k)$ and $r_j = \sum_{k=1}^j \rho(e_k)$ be the prefix sums of the weights. Observe that $f = \ell_m + 1$. We store a stack $\gamma = ((e_1, \ell_1, r_1), (e_2, \ell_2, r_2), \dots, (e_m, \ell_m, r_m))$, implemented as an array. Given $i \in [1, m]$, one can pop all elements at positions $i+1, \dots, m$ in constant time. We store the set of distinct prefix sums $L = \{\ell_j \mid 0 \leq j \leq m\}$ in a dynamic predecessor data structures from Theorem 14 where a prefix sum ℓ is stored together with the maximal index j such that $\ell = \ell_j$. Similarly $R = \{r_j \mid 0 \leq j \leq m\}$ is stored in a predecessor data structure.

For $\text{setfinger}(f)$ we compute an arbitrary accelerated path from S to $S[f]$, say only using only short steps, and set up the list γ and the predecessor data structures for L and R in time $\mathcal{O}(\log N)$. For $\text{movefinger}(i)$ we can assume that $f - i = d > 0$ since the data structures are left-right symmetric. By a predecessor query on L we can find the unique index j with $\ell_j < i \leq \ell_{j+1}$. Then we restrict γ to its prefix of length j , and perform $\text{split}(L, \ell_j)$ and $\text{split}(R, r_j)$, all in constant time. Using fringe access we can compute an accelerated path

π' from A to $S[i] = A[i']$ where $i' = i - \ell_j$: If e_{j+1} is a short edge we take a short step and then use Proposition 16 for the remaining path. If e_{j+1} is a left or right long edge in a forest $F \in \mathcal{F}$ we take a *left* long step, followed by a short step, and then use Proposition 16 for the remaining path. Finally, we update the stack γ and the prefix sums in L and R in time $\mathcal{O}(|\pi'|)$. This concludes the proof of Theorem 3.

We leave it as an open question whether there exists a linear space finger search data structure, supporting $\text{access}(i)$ and $\text{movefinger}(i)$ in $\mathcal{O}(\log d)$ time. For path balanced SLPs such a solution does exist.

► **Theorem 17.** *Given an (α, β) -path balanced SLP of size g for a string of length N , one can support $\text{setfinger}(i)$ in $\mathcal{O}(\log N)$ time, and $\text{access}(i)$ and $\text{movefinger}(i)$ in $\mathcal{O}(\log d)$ time, where d is the distance between i and the current finger position, after $\mathcal{O}(g)$ preprocessing time and space.*

References

- 1 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-Grained Complexity of Analyzing Compressed Data: Quantifying Improvements over Decompress-and-Solve. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.26.
- 2 Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger Search in Grammar-Compressed Strings. *Theory Comput. Syst.*, 62(8):1715–1735, 2018. doi:10.1007/s00224-017-9839-9.
- 3 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random Access to Grammar-Compressed Strings and Trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.
- 4 Gerth Stølting Brodal. Finger Search Trees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch11.
- 5 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
- 6 Martin Farach and S. Muthukrishnan. Perfect Hashing for Strings: Formalization and Algorithms. In Daniel S. Hirschberg and Eugene W. Myers, editors, *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, California, USA, June 10-12, 1996, Proceedings*, volume 1075 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 1996. doi:10.1007/3-540-61258-0_11.
- 7 Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A Faster Grammar-Based Self-index. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 6th International Conference, LATA 2012, A Coruña, Spain, March 5-9, 2012. Proceedings*, volume 7183 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2012. doi:10.1007/978-3-642-28332-1_21.
- 8 Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-Based Self-indexing with Faster Pattern Matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.
- 9 Moses Ganardi. Compression by Contracting Straight-Line Programs. *CoRR*, abs/2107.00446, 2021. URL: <https://arxiv.org/abs/2107.00446>.
- 10 Moses Ganardi, Artur Jeż, and Markus Lohrey. Balancing Straight-Line Programs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*

- 2019, Baltimore, Maryland, USA, November 9-12, 2019, pages 1169–1183. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00073.
- 11 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-Based Compression of Unranked Trees. *Theory Comput. Syst.*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.
 - 12 Leszek Gasieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant. Real-Time Traversal in Grammar-Based Compressed Files. In *2005 Data Compression Conference (DCC 2005), 29-31 March 2005, Snowbird, UT, USA*, page 458. IEEE Computer Society, 2005. doi:10.1109/DCC.2005.78.
 - 13 Pawel Gawrychowski. Pattern Matching in Lempel-Ziv Compressed Strings: Fast, Simple, and Deterministic. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2011. doi:10.1007/978-3-642-23719-5_36.
 - 14 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted Ancestors in Suffix Trees. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wrocław, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.
 - 15 Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 565–574. SIAM, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283444>.
 - 16 N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.
 - 17 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.
 - 18 Markus Lohrey, Sebastian Maneth, and Carl Philipp Reh. Constant-Time Tree Traversal and Subtree Equality Check for Grammar-Compressed Trees. *Algorithmica*, 80(7):2082–2105, 2018. doi:10.1007/s00453-017-0331-3.
 - 19 Craig G. Nevill-Manning and Ian H. Witten. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997. doi:10.1613/jair.374.
 - 20 Mihai Patrascu and Mikkel Thorup. Dynamic Integer Sets with Optimal Rank, Select, and Predecessor Search. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 166–175. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.26.
 - 21 Carl Philipp Reh and Kurt Sieber. Navigating Forest Straight-Line Programs in Constant Time. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2020. doi:10.1007/978-3-030-59212-7_2.
 - 22 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.
 - 23 Robert Endre Tarjan. A Class of Algorithms which Require Nonlinear Time to Maintain Disjoint Sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.
 - 24 Terry A. Welch. A Technique for High-Performance Data Compression. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 17(6):8–19, 1984. doi:10.1109/MC.1984.1659158.
 - 25 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Space Efficient Two-Dimensional Orthogonal Colored Range Counting

Younan Gao ✉

Faculty of Computer Science, Dalhousie University, Halifax, Canada

Meng He ✉

Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

In the two-dimensional orthogonal colored range counting problem, we preprocess a set, P , of n colored points on the plane, such that given an orthogonal query rectangle, the number of distinct colors of the points contained in this rectangle can be computed efficiently. For this problem, we design three new solutions, and the bounds of each can be expressed in some form of time-space tradeoff. By setting appropriate parameter values for these solutions, we can achieve new specific results with (the space costs are in words and ϵ is an arbitrary constant in $(0, 1)$):

- $O(n \lg^3 n)$ space and $O(\sqrt{n} \lg^{5/2} n \lg \lg n)$ query time;
- $O(n \lg^2 n)$ space and $O(\sqrt{n} \lg^{4+\epsilon} n)$ query time;
- $O(n \frac{\lg^2 n}{\lg \lg n})$ space and $O(\sqrt{n} \lg^{5+\epsilon} n)$ query time;
- $O(n \lg n)$ space and $O(n^{1/2+\epsilon})$ query time.

A known conditional lower bound to this problem based on Boolean matrix multiplication gives some evidence on the difficulty of achieving near-linear space solutions with query time better than \sqrt{n} by more than a polylogarithmic factor using purely combinatorial approaches. Thus the time and space bounds in all these results are efficient. Previously, among solutions with similar query times, the most space-efficient solution uses $O(n \lg^4 n)$ space to answer queries in $O(\sqrt{n} \lg^8 n)$ time (SIAM. J. Comp. 2008). Thus the new results listed above all achieve improvements in space efficiency, while all but the last result achieve speed-up in query time as well.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases 2D Colored orthogonal range counting, stabbing queries, geometric data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.46

Related Version Full Version: <https://arxiv.org/pdf/2107.02787.pdf>

Funding This work was supported by NSERC of Canada.

1 Introduction

In computational geometry, there have been extensive studies on problems over points associated with information represented as colors [13, 14, 20, 19, 21, 11, 23, 10, 12, 5, 3, 16, 24]. Among them, the *2D orthogonal range counting* query problem is one of the most fundamental. In this problem, we preprocess a set, P , of n points on the plane, each colored in one of C different colors, such that given an orthogonal query rectangle, the number of distinct colors of the points contained in this rectangle can be computed efficiently.

This problem is important in both theory and practice. Theoretically, it has connections to matrix multiplication: The ability to answer m colored range counting queries offline over n points on the plane in $o(\min\{n, m\}^{\omega/2})$ time, where ω is the best current exponent of the running time of matrix multiplication, would yield a faster algorithm for Boolean matrix multiplication [19]. In practice, the records in database systems and many other applications are often associated with categorical information which can be modeled as colors. The Structured Programming Language (SQL) thus provides keywords such as `DISTINCT`



© Younan Gao and Meng He;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 46; pp. 46:1–46:17



Leibniz International Proceedings in Informatics

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

and GROUP BY to compute information about the distinct categories of the records within a query range, which can be modeled using colored range query problems, and these queries have also been used in database query optimization [6].

One challenge in solving the 2D orthogonal range counting problem is that the queries are not easily decomposable: if we partition the query range into two or more subranges, we cannot simply obtain the number of distinct colors in the query range by adding up the number of distinct colors in each subrange. Furthermore, the conditional lower bound based on matrix multiplication as described above gives theoretical evidence on the hardness of this problem. Indeed, if polylogarithmic query times are desired, the solution with the best space efficiency [22] uses $O(n^2 \lg n / \lg \lg n)$ words of space to answer queries in $O((\lg n / \lg \lg n)^2)$ time. There is a big gap between the complexities of this solution to those of the optimal solution to *2D orthogonal range counting* for which we do not have color information and are only interested in computing the number of points in a 2D orthogonal query range. The latter can be solved in merely linear space and $O(\lg n / \lg \lg n)$ query time [17].

Applications that process a significant amount of data would typically require structures whose space costs are much lower than quadratic. As the running time of the best known combinatorial algorithm of multiplying two $n \times n$ Boolean matrices is $\Theta(n^3 / \text{polylog}(n))$ [1, 2, 26], the conditional lower bound of 2D orthogonal colored range counting implies that no solution can simultaneously have preprocessing time better than $\Omega(n^{3/2})$ and query time better than $\Omega(\sqrt{n})$, by purely combinatorial methods¹ with current knowledge, save for polylogarithmic speed-ups. To match this query time within polylogarithmic factors, the most space-efficient solution uses $O(n \lg^4 n)$ words of space to answer queries in $O(\sqrt{n} \lg^8 n)$ time [19]. Despite this breakthrough, the exponents in the polylogarithmic factors in the time and space bounds leave much room for potential improvements. Hence, in this paper, we aim at decreasing these polylogarithmic factors in both space and time costs, to design solutions that are more desirable for applications that manage large data sets.

Previous Work. Gupta et al. [13] showed how to reduce the orthogonal colored range searching problem in 1D to orthogonal range searching over uncolored points in 2D, thus achieving a linear-space solution with $O(\lg n / \lg \lg n)$ query time. Later, the query time was improved to $O(\lg C / \lg \lg n)$ by Nekrich [23], where C is the number of colors.

To solve 2D colored orthogonal range counting, Gupta et al. [13] used persistent data structures to extend their 1D solution to 2D and designed a data structure of $O(n^2 \lg^2 n)$ words that supports queries in $O(\lg^2 n)$ time. Kaplan et al. [19] achieved the same bounds by decomposing the input points into disjoint boxes in 3D and reduced the problem to 3D stabbing counting queries. Recently, Munro et al. [22] showed that 3D 3-sided colored range counting can be answered in $O((\lg n / \lg \lg n)^2)$ time using a data structure of $O(n(\lg n / \lg \lg n))$ words of space, which implies a solution to 2D 3-sided colored range counting with the same time and space bound. For each distinct x -coordinate x_i of the points in the point set, if we use the strategy in [13, 19] to build a data structure supporting 2D 3-sided queries upon the points whose x -coordinates are greater than or equal to x_i , then this set of data structures constructed can be used to answer a 4-sided query. This yields a solution to the 2D orthogonal colored range counting problem with $O(n^2 \lg n / \lg \lg n)$ words of space and $O((\lg n / \lg \lg n)^2)$ query time. Kaplan et al. [19] further showed how to achieve time-space tradeoffs by designing a solution with $O(X \lg^7 n)$ query time that uses $O((\frac{n}{X})^2 \lg^6 n + n \lg^4 n)$

¹ When algebraic approaches are allowed, $\omega \approx 2.3727$ [25], implying that the preprocessing time and the query time cannot be simultaneously less than $\Omega(n^{1.18635})$ and $\Omega(n^{0.18635})$, respectively.

words of space. Setting $X = \sqrt{n} \lg n$ minimizes space usage, achieving an $O(n \lg^4 n)$ -word solution with $O(\sqrt{n} \lg^8 n)$ query time. When only linear space is allowed, Grossi and Vind [11] showed how to answer a query in $O(n / \text{polylog}(n))$ time. Though not explicitly stated anywhere, by combining an approach that Kaplan et al. [19] presented for dimensions of 3 or higher (which also works for 2D) and a linear space solution to 2D orthogonal range emptiness [4], the query time can be improved to $O(n^{3/4} \lg^\epsilon n)$ for any constant ϵ using a linear space structure. Finally, Kaplan et al. also considered the offline version of this problem and showed that n 2D orthogonal colored range counting queries can be answered in $O(n^{1.408})$ time.

Researchers have also studied approximate colored range counting problem. In 1D, El-Zein et al. [10] designed a succinct data structure to answer a $(1 + \epsilon)$ -approximate colored counting query in constant time. In 2D, Rahul [24] provided a reduction from $(1 + \epsilon)$ -approximate orthogonal colored range counting to 2D colored orthogonal range reporting which reports the number of distinct colors in a 2D orthogonal query range. Based on this, they gave an $O(n \lg n)$ -word data structure with $O(\lg n)$ query time.

The orthogonal colored range counting problem has also been studied in higher dimensions [21, 19, 24, 16]. Furthermore, He and Kazi [16] generalized it to categorical path counting by replacing the first dimension with tree topology. One of their solutions to a path query problem generalized from 2D orthogonal colored range counting also uses linear space and provides $O(n^{3/4} \lg^\epsilon n)$ query time. We end this brief survey by commenting that, after a long series of work [18, 13, 19, 23, 11, 12], Chan and Nekrich [5] solved the related 2D orthogonal range reporting problem in $O(n \lg^{3/4+\epsilon} n)$ words of space and $O(\lg \lg n + k)$ query time for points in rank space, where k is output size. This almost matches the bounds of the optimal solution to (uncolored) 2D orthogonal range reporting over points in rank space, which uses $O(n \lg^\epsilon n)$ words to answer queries in $O(\lg \lg n + k)$ time.

Our Results. Under the word RAM model, we present three results, all in the form of time-space tradeoffs, for two-dimensional orthogonal colored range counting. Specifically, for an integer parameter $X \in [1, n]$, we propose solutions (all space costs are in words):

- with $O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$ space and $O(\lg^4 n + X \lg^2 n \lg \lg n)$ query time; setting $X = \sqrt{n} \lg n$ achieves $O(n \lg^3 n)$ space and $O(\sqrt{n} \lg^{5/2} \lg \lg n)$ query time;
- with $O((\frac{n}{X})^2 \lg^4 n + n \lg^2 n)$ space and $O(\lg^6 n + X \lg^{3+\epsilon} n)$ query time for any constant $\epsilon \in (0, 1)$; setting $X = \sqrt{n} \lg n$ achieves $O(n \lg^2 n)$ space and $O(\sqrt{n} \lg^{4+\epsilon})$ query time;
- with $O((\frac{n}{X})^2 \lg^2 n \cdot \log_\lambda^2 n + n \lg n \cdot \log_\lambda n)$ space and $O(\lambda^2 \cdot \lg^6 n \cdot \log_\lambda^2 n + X \cdot \lg^{3+\epsilon} n \cdot \lambda \log_\lambda n)$ query time for an integer parameter $\lambda \in [2, n]$; setting $X = \sqrt{n \lg n \log_\lambda n}$ and $\lambda = \lg^\epsilon n$ achieves $O(n \frac{\lg^2 n}{\lg \lg n})$ space and $O(\sqrt{n} \lg^{5+\epsilon'})$ query time for any $\epsilon' > 2\epsilon$, while setting $X = \sqrt{n \lg n}$ and $\lambda = n^{\epsilon/5}$ achieves $O(n \lg n)$ space and $O(n^{1/2+\epsilon})$ query time.

When presenting each result, we also showed the bounds of the most space-efficient tradeoff that can be achieved by setting appropriate parameter values. The conditional lower bound based on Boolean matrix multiplication which we discussed before gives some evidence on the difficulty of achieving query time better than \sqrt{n} by more than a polylogarithmic factor using combinatorial approaches without increasing these space costs polynomially.

When comparing to previous results, note that only the time-space tradeoff presented by Kaplan et al. [19] could possibly achieve near-linear space and $O(\sqrt{n} \text{polylog}(n))$ query time. More specifically, their solution uses $O((\frac{n}{X})^2 \lg^6 n + n \lg^4 n)$ words of space to achieve $O(X \lg^7 n)$ query time. The most space-efficient tradeoff that could be obtained from it is an $O(n \lg^4 n)$ -word structure with $O(\sqrt{n} \lg^8 n)$ query time. Thus we indeed achieve the goal of improving the polylogarithmic terms in both their time and space costs significantly.

■ **Table 1** Bounds of 2D orthogonal colored range counting structures. The results in the form of time-space tradeoffs are listed in the top portion, in which X and λ are integer parameters in $[1, n]$ and $[2, n]$, respectively. The bottom portion presents results with specific bounds, among which marked with a \dagger are those obtained from the top portion by setting appropriate parameters values.

Source	Model	Query Time	Space Usage in Words
[19]	PM	$O(X \lg^7 n)$	$O((\frac{n}{X})^2 \lg^6 n + n \lg^4 n)$
Cor. 6	PM	$O(\lg^5 n + X \lg^3 n)$	$O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$
Thm. 5	RAM	$O(\lg^4 n + X \lg^2 n \lg \lg n)$	$O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$
Thm. 9	RAM	$O(\lg^6 n + X \lg^{3+\epsilon} n)$	$O((\frac{n}{X})^2 \lg^4 n + n \lg^2 n)$
Thm. 12	RAM	$O(\lambda^2 \lg^6 n \log_\lambda^2 n + X \lg^{3+\epsilon} n \lambda \log_\lambda n)$	$O((\frac{n}{X})^2 \lg^2 n \log_\lambda^2 n + n \lg n \log_\lambda n)$
[13, 19]	PM	$O(\lg^2 n)$	$O(n^2 \lg^2 n)$
[22]	RAM	$O((\lg n / \lg \lg n)^2)$	$O(n^2 \lg n / \lg \lg n)$
[19] [†]	PM	$O(\sqrt{n} \lg^8 n)$	$O(n \lg^4 n)$
Cor. 6 [†]	PM	$O(\sqrt{n} \lg^{7/2} n)$	$O(n \lg^3 n)$
Thm. 5 [†]	RAM	$O(\sqrt{n} \lg^{5/2} n \lg \lg n)$	$O(n \lg^3 n)$
Thm. 9 [†]	RAM	$O(\sqrt{n} \lg^{4+\epsilon} n)$	$O(n \lg^2 n)$
Thm. 12 [†]	RAM	$O(\sqrt{n} \lg^{5+\epsilon} n)$	$O(n \frac{\lg^2 n}{\lg \lg n})$
Thm. 12 [†]	RAM	$O(n^{1/2+\epsilon})$	$O(n \lg n)$
[19]	PM	$O(n^{3/4} \lg n)$	$O(n \lg n)$
[11]	RAM	$O(n / \text{polylog}(n))$	$O(n)$
[19, 4]	RAM	$O(n^{3/4} \lg^\epsilon n)$	$O(n)$

It is worthwhile to mention that the result of Kaplan et al. can work under the pointer machine (PM) model. Thus, for an absolutely fair comparison, we show how our first result can be adapted to the same model of computation to achieve $O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$ space and $O(\lg^5 n + X \lg^3 n)$ query time. Thus under PM, we have an $O(n \lg^3 n)$ -word structure with $O(\sqrt{n} \lg^{7/2} \lg n)$ query time. This is still a significant improvement over previous similar results. In the rest of the paper, however, we assume the word RAM model of computation unless otherwise specified, since most of our results are designed under it. See Table 1 for a comparison of our results to all previous results.

To achieve these results, we use the standard technique of decomposing a 4-sided query range to two 3-sided subranges with a range tree. Then the answer can be obtained by adding up the numbers of distinct colors assigned to points in each subrange and then subtracting the number of distinct colors that exist in both. We still use an approach of Kaplan et al. to reduce 2D 3-sided colored range counting to 3D stabbing queries over a set of boxes. What is new is our scheme of achieving time-space tradeoffs when computing the number of colors that exist in both subranges. Based on a parameter, we selectively precompute the sizes of the intersections between pairs of colors sets, each of which corresponds to a prefix of a certain box list somewhere in the stabbing query structures. Compared to the scheme of Kaplan et al. for the same purpose, ours gives more flexibility in the design of the 3D stabbing query structures that could work with the scheme. This extra flexibility further allows us to use and design different stabbing query structures to achieve new results.

2 Preliminaries

In this section, we introduce some notation and previous results used in our paper.

Notation. Throughout this paper, we assume points are in general positions unless otherwise specified. In three-dimensional space, we call a box B *canonical* if it is defined in the form of $[x_1, +\infty) \times [y_1, y_2) \times [z_1, z_2)$, where $x_1, y_1, z_1 \in \mathbb{R}$ and $y_2, z_2 \in \mathbb{R} \cup \{+\infty\}$. We use $B.x_1$ to refer to the lower bound of the x -range of B , $B.y_1$ and $B.y_2$ to respectively refer to the lower and upper bounds of the y -range of B , and so on. Let $(p.x, p.y, p.z)$ denote the coordinates of a point p . We say a point $q \in \mathbb{R}^3$ *dominates* another point $p \in \mathbb{R}^3$, if $q.x \geq p.x$, $q.y \geq p.y$ and $q.z \geq p.z$ hold simultaneously.

2D Orthogonal Colored Range Emptiness. An *orthogonal range emptiness* query determines whether an axis-aligned query rectangle contains at least one point in the point set P . Observe that a solution to this query problem directly leads to a solution to the colored version of this problem called *orthogonal colored range emptiness*, in which each point in P is colored in one of C different colors, and given a color c and an axis-aligned rectangle, the query asks whether the query range contains at least one point colored in c . The reduction works as follows: For each color $1 \leq c \leq C$, let P_c denote the subset of P containing all points colored in c . If we construct an orthogonal range emptiness structure over P_c for each color c , then we can answer an orthogonal colored range emptiness query by querying the structure constructed over the points with the query color. The following lemma thus directly follows from the work of Chan et al. [4] on range emptiness:

► **Lemma 1** ([4]). *Given n colored points in 2-dimensional rank space, there is a data structure of $nf(n)$ words that answers 2D orthogonal colored range emptiness queries in $g(n)$ time, where*

- a) $f(n) = O(1)$ and $g(n) = O(\lg^\epsilon n)$ for any constant $\epsilon > 0$; or
- b) $f(n) = O(\lg \lg n)$ and $g(n) = O(\lg \lg n)$.

Orthogonal Stabbing Queries over 3D Canonical Boxes. In the *3D stabbing counting* problem, we preprocess a set of 3D boxes, such that, given a query point q , we can compute the number of boxes containing q efficiently, while in the *3D stabbing reporting* query problem, we report these boxes. In both our solution and the solution of Kaplan et al. [19], we use data structures for this problem in a special case in which each box is a canonical box.

Here we introduce the data structure of Kaplan et al. [19] that solves the stabbing query problems over a set of n canonical boxes in 3D, as our first solution to the colored range counting problem augments this data structure and uses it as a component. Their data structure consists of two layers of segment trees. The structure at the top layer is a segment tree constructed over the z -coordinates of the boxes. More precisely, we project each box onto the z -axis to obtain an interval, and the segment tree is constructed over all these intervals. A box is assigned to a node in this tree if its corresponding interval on the z -axis is *associated* with this node. (Recall that as a segment tree, the leaves correspond to the elementary intervals induced by the endpoints of the intervals projected by each box onto the z -axis; and each internal node v of the tree corresponds to an interval $I_z(v)$ that are the union of elementary intervals of the leaves in the subtree rooted at v . A box is assigned to a node v in this tree if its corresponding interval on the z -axis covers the interval $I_z(v)$ but does not cover the interval $I_z(u)$, where u is the parent node of v .) For each node v in the top-layer segment tree, we further construct a segment tree over the projections of the

boxes assigned to v on the y -axis, and the segment trees constructed for all the nodes of the top-layer tree form the bottom-layer structure. Finally, for each node, v' , of a segment tree at the bottom layer, we construct a list, $sList(v')$, of the boxes assigned to v' , sorted by their x_1 -coordinates, i.e. the left endpoints of their projections on the x -axis; recall that the right endpoint is always $+\infty$ by the definition of canonical boxes. The lists of boxes associated with the nodes in the same bottom-level segment tree are linked together in a fractional cascading [8] data structure to facilitate the search among x_1 -coordinates of the boxes in each list.

With these data structures, the query algorithm first performs a search in the top-layer segment tree to locate the $O(\lg n)$ nodes whose associated intervals (which are projections of boxes on the z -axis) contain the z -coordinate of the query point q . For each of these nodes, we then query the bottom-layer segment tree constructed for it to find the boxes whose projections on the yz -plane contain $(q.y, q.z)$. The searches with this layer of structures end at $O(\lg^2 n)$ nodes of the bottom-layer trees, whose $sList$'s are disjoint. Finally, a binary search at the $sList$ of each of these nodes, sped up by fractional cascading, gives us the answer. Thus:

► **Lemma 2** ([19]). *Given a set of n canonical boxes in three dimension, the above data structure occupies $O(n \lg^2 n)$ words and answers stabbing counting queries in $O(\lg^2 n)$ time and stabbing reporting queries in $O(\lg^2 n + k)$ time, where k denotes the number of boxes reported. Furthermore, the output of the reporting query is the union of $O(\lg^2 n)$ disjoint subsets, each containing the boxes stored in a nonempty prefix of a bottom-layer sorted list. The preprocessing time is $O(n \lg^2 n)$.*

Reducing 2D 3-Sided Colored Range Counting to 3D Orthogonal Stabbing Counting over Canonical Boxes. A key technique used in both the solutions of Kaplan et al. [19] and our solutions is a reduction from 2D 3-sided colored range counting queries, in which each query range is of the form $[a, b] \times [c, +\infty)$ for some $a, b, c \in \mathbb{R}$, to 3D orthogonal stabbing queries over canonical boxes². The reduction is performed in two steps. First we reduce the 2D 3-sided colored range counting query problem to the 3D dominance colored range counting problem, in which we preprocess a set, P , of colored points in \mathbb{R}^3 , such that given a query point q in \mathbb{R}^3 , one can report the number of distinct colors in $P \cap (-\infty, q.x] \times (-\infty, q.y] \times (-\infty, q.z]$ efficiently. This reduction works as follows: For each point, $p = (p.x, p.y)$, we create a point, $p' = (-p.x, p.x, -p.y)$, in \mathbb{R}^3 , and assign it with the color of p . Then a 2D 3-sided colored range counting query over the original points in which the query range is $[a, b] \times [c, +\infty)$ can be answered by performing a 3D dominance colored range query over the created points, using $(-\infty, -a] \times (-\infty, b] \times (-\infty, -c]$ as the query range.

To further reduce the 3D dominance colored range counting problem to 3D orthogonal stabbing counting over canonical boxes, we need some additional notation: Given a point p in 3D, let Q_p^+ denote region $[p.x, +\infty) \times [p.y, +\infty) \times [p.z, +\infty)$. Furthermore, given a point set A , let $U(A)$ denote the region of $\cup_{p \in A} Q_p^+$. Then the following lemma is crucial:

► **Lemma 3** ([19]). *Given a set, A , of n points in three-dimensional space, a set of $O(n)$ pairwise disjoint 3D canonical boxes can be computed in $O(n \lg^2 n)$ time such that the union of these boxes is the region $U(A)$.*

² Later we sometimes deal with query ranges of the form $[a, b] \times (\infty, d]$ for some $a, b, d \in \mathbb{R}$, and a similar reduction also works.

Originally Kaplan et al. [19, Theorem 2.1] proved the above lemma for d -dimensional space where $d \geq 1$; for a general d , the number of boxes required to cover $U(A)$ is $O(n^{\lfloor d/2 \rfloor})$.

With this lemma, the reduction works as follows. Let P denote the input colored point set of the 3D dominance colored range counting problem, and let C denote the number of colors. Then, for each color $1 \leq c \leq C$, we apply Lemma 3 to partition $U(P_c)$, where P_c is the set of all the points in P that are colored c , into a set, B_c , of $O(|U(P_c)|)$ disjoint 3D canonical boxes. We then construct a 3D stabbing counting structure over $B = \cup_{c=1}^C B_c$. Note that this data structure is constructed over $O(|P|)$ canonical boxes, as $\sum_{c=1}^C |B_c| = \sum_{c=1}^C O(|P_c|) = O(|P|)$. To answer a 3D dominance colored range counting query over P , for which the query range is the region dominated by a point q , observe that, if q dominates at least one point in P_c , then it must be located with $U(P_c)$. Then, since $U(P_c)$ is partitioned into the boxes in B_c , we conclude that q dominates at least one point in P_c iff q is contained in a box in B_c . Furthermore, since the boxes in B_c are pairwise disjoint, q is either contained in exactly one box in B_c or outside $U(P_c)$. Hence, the number of distinct colors in the region dominated by q is equal to the number of boxes in B that contains q , which can be computed by performing a stabbing counting query in B using q as the query point.

3 A New Framework of Achieving Time-Space Tradeoffs

We present three new solutions to 2D orthogonal colored range counting in this section and Section 4. They follow the same framework, of which we give an overview in Section 3.1. One key component of this framework is a novel scheme of computing the sizes of the intersections between the color sets assigned to different subsets of points that lie within the query range; Section 3.2 describes this scheme and shows how to combine it with Lemma 2 to immediately achieve a new time-space tradeoff for 2D orthogonal colored range counting.

3.1 Overview of the Data Structure Framework

Let P denote a set of n points on the plane, each assigned a color identified by an integer in $[1, C]$. To support orthogonal colored range counting over P , we construct a binary range tree T over the y -coordinates of the points in P such that each leaf of T stores a point of P , and, from left to right, the points stored in the leaves are increasingly sorted by y -coordinate. For each internal node v of T , we construct the following data structures:

- A list $P(v)$ containing the points stored at the leaf descendants of v , sorted by x -coordinate;
- A list $P_y(v)$ containing the sorted list of y -coordinates of the points in $P(v)$;
- 2D 3-sided colored range counting structures, $S(v_l)$ and, $S(v_r)$, constructed over $P(v_l)$ and $P(v_r)$ respectively, where v_l and v_r are respectively the left and right children of v ($S(v_l)$ requires query ranges to open at the top while $S(v_r)$ requires them to open at the bottom; the exact data structures used will be selected later for different tradeoffs);
- 2D orthogonal colored range emptiness query structures, $E(v_l)$, and, $E(v_r)$, constructed over $\hat{P}(v_l)$ and $\hat{P}(v_r)$, respectively (using Lemma 1), where $\hat{P}(v_l)$ and $\hat{P}(v_r)$ are the point set in rank space converted from $P(v_l)$ and $P(v_r)$.

Let $Q = [a, b] \times [c, d]$ be the query rectangle. Given an internal node v , let $C_Q(v)$ denote the set of distinct colors assigned to points in $P(v) \cap Q$. The query algorithm first locates the lowest common ancestor u of the c -th and d -th leaves of T . As all the points from P that are in the query range Q must be in $P(u)$, $|C_Q(u)|$ is the answer to the query. To compute $|C_Q(u)|$, let u_l and u_r denote the left and right children of u , respectively. By the exclusion-inclusion principle, we know that $|C_Q(u)| = |C_Q(u_l)| + |C_Q(u_r)| - |C_Q(u_l) \cap C_Q(u_r)|$.

Among the terms on the right hand side of this equation, $|C_Q(u_l)|$ and $|C_Q(u_r)|$ can be computed by performing 2D 3-sided colored range counting queries over $S(v_l)$ and $S(v_r)$, using $[a, b] \times [c, +\infty)$ and $[a, b] \times (-\infty, d]$ as query ranges, respectively. What remains is to compute $|C_Q(u_l) \cap C_Q(u_r)|$.

This idea of decomposing a 4-sided query range into two 3-sided query ranges has been used before for both 2D orthogonal colored range reporting [12] and counting [19]. Furthermore, to support 2D 3-sided colored range counting, we apply the reduction of Kaplan et al. [19] (summarized in Section 2 of our paper) to reduce it to the 3D stabbing query problems over canonical boxes. Thus, the techniques summarized so far have been used in previous work (without the construction of range emptiness structures). What is new is our scheme of achieving time-space tradeoffs when computing $|C_Q(u_l) \cap C_Q(u_r)|$; it gives us more *flexibility* in the design of 3D stabbing query structures, thus allowing us to achieve new results.

Here we describe the conditions that a 3D stabbing query structure must meet so that we can combine it with our scheme of computing $|C_Q(u_l) \cap C_Q(u_r)|$, while deferring the details of the latter to Section 3.2. The stabbing query structure consists of multiple layers of trees of some kind. The top-layer tree is constructed over the entire set of canonical boxes, and each of its nodes is assigned a subset of boxes. The second layer consists of a set of trees, each constructed over the boxes assigned to a node in the top-layer tree, and so on. Thus each bottom-layer tree node is also assigned a list of boxes in a certain order (e.g., the *sList* in the data structure for Lemma 2). The query algorithm locates a set S of bottom-layer tree nodes. For each node $v \in S$, there exists a nonempty prefix of the box list assigned to v , such that such prefixes over all the nodes in S form a partition of the set of boxes containing the query point q . Furthermore, the size of each such prefix can be computed efficiently, and each box in such a prefix can also be reported efficiently.

Clearly Lemma 2 satisfies these conditions and can be used in our framework. On the contrary, even though Kaplan et al. proved this lemma and used it successfully in their $O(n^2 \lg^2 n)$ space solution, they can not directly use it with their scheme of achieving time-space tradeoffs. Instead, they expand this structure with a third layer which is a segment tree constructed over x -coordinates of the boxes, increasing both time and space costs. Recall that in their scheme the set of boxes containing the query point are also decomposed into a fixed number of subsets. The reason Kaplan et al. cannot directly apply Lemma 2 is that their scheme requires each of the decomposed subsets to be equal to the entire set of boxes stored in a bottom-layer tree node, while our scheme allows each subset to be part of such a box set. Hence, this extra flexibility allows us to use Lemma 2 and alternative 3D stabbing query structures to be designed later (in Section 4) in our framework.

3.2 Computing Intersections between Color Sets

We now introduce our new scheme of computing $|C_Q(u_l) \cap C_Q(u_r)|$, and combine it with the stabbing query structure from Lemma 2 to achieve a new time-space tradeoff for 2D orthogonal colored range counting. Since our scheme works with some other stabbing query structures, we describe it assuming a stabbing query structure satisfying the conditions described in Section 3.1 is used. To understand this scheme more easily, it may be advisable for readers to think about how it applies to the stabbing query structure of Lemma 2.

Recall that, the problems of computing $|C_Q(u_l)|$ and $|C_Q(u_r)|$ have each been reduced to a 3D stabbing query. Furthermore, for each stabbing query, all reported boxes are distributed into a number of disjoint sets; the boxes in each such set form a nonempty prefix of the box list assigned to a node of the bottom-layer tree (henceforth we call each such box list a *bottom list* for convenience). Then, for the stabbing query performed to compute $|C_Q(u_l)|$, we

define D_Q to be a set in which each element is a such a disjoint set, and all these disjoint sets (whose union form the set of reported boxes) are elements of D_Q . U_Q is defined in a similar way for $|C_Q(u_r)|$. Thus, if we use the data structure for Lemma 2 to answer these stabbing queries, both $|D_Q|$ and $|U_Q|$ will be upper bounded by $O(\lg^2 n)$. As shown in Section 2, when reducing 2D 3-sided colored counting to 3D stabbing queries over canonical boxes, we guarantee that, each canonical box is part of the region $U(P_c)$ for each color c ; we call this color the color of this canonical box, and explicitly store with each box its color. Furthermore, for each color $1 \leq c \leq C$, at most one canonical box colored in c contains the query point, which implies that each box in $\cup_{s \in D_Q} s$ (resp. $\cup_{t \in U_Q} t$) has a distinct color. For each set $s \in D_Q$ and $t \in U_Q$, let $C(s)$ and $C(t)$ denote the set of colors associated with the boxes in s and t , respectively. Then we have $|C_Q(u_l)| = \sum_{s \in D_Q} |C(s)|$, $|C_Q(u_r)| = \sum_{t \in U_Q} |C(t)|$, and $|C_Q(u_l) \cap C_Q(u_r)| = \sum_{s \in D_Q, t \in U_Q} |C(s) \cap C(t)|$.

It now remains to show how to compute $\sum_{s \in D_Q, t \in U_Q} |C(s) \cap C(t)|$, for which more preprocessing is required. For each node v in the binary range tree T , we construct a matrix $M(v)$ as follows: Let $X \in [1, n]$ be a parameter to be chosen later. If the length, m , of a bottom list in the stabbing query structure $S(v_l)$ or $S(v_r)$ is greater than X , we divide the list into $\lceil m/X \rceil$ blocks, such that each block, with the possible exception of the last block, is of length X . If $m \leq X$, then the entire list is a single block. If a block is of length X , we call it a *full block*. Let $b_l(v)$ and $b_r(v)$ denote the total numbers of full blocks over all bottom lists in $S(v_l)$ and $S(v_r)$, respectively. Then $M(v)$ is a $b_l(v) \times b_r(v)$ matrix, in which each row (or column) corresponds to a nonempty prefix of a bottom list in $S(v_l)$ (or $S(v_r)$) that ends with the last entry of a full block, and each entry $M[i, j]$ stores the number of colors that exist in both the set of colors assigned to the boxes in the prefix corresponding to row i and the set of colors assigned to the boxes in the prefix corresponding to column j . To bound the size of $M(v)$, we define the *duplication factor*, $\delta(n)$, of a stabbing query structure that satisfies the conditions in Section 3.1 to be the maximum number of bottom lists that any canonical box can be contained in. For example, the duplication factor of the structure for Lemma 2 is $O(\lg^2 n)$. (To see this, observe that in a segment tree constructed over n intervals, each interval may be stored in $O(\lg n)$ tree nodes. Since segment trees are used in both layers of the data structure for Lemma 2, each box can be stored in $O(\lg^2 n)$ different bottom lists.) Since each full block contains X boxes, the numbers of full blocks in the bottom lists of $S(v_l)$ and $S(v_r)$ are at most $\delta(n)|P(v_l)|/X$ and $\delta(n)|P(v_r)|/X$, respectively. Therefore, $M(v)$ occupies $O(\delta(n)^2 |P(v_l)| |P(v_r)| / X^2) = O((\delta(n) |P(v)| / X)^2)$ words.

With these matrices, the computation of $\sum_{s \in D_Q, t \in U_Q} |C(s) \cap C(t)|$ can proceed as follows. Since each set $s \in D_Q$ (resp. $t \in U_Q$) occupies a prefix of a bottom list, s (resp. t) can be split into two parts: s_h (resp. t_h) which is the (possibly empty) prefix of s (resp. t) that consists of all the full blocks entirely contained in s (resp. t), and s_l (resp. t_l) which contains the remaining entries of s (resp. t). Thus we have $C(s_h) \cup C(s_l) = C(s)$ and $C(t_h) \cup C(t_l) = C(t)$. Since no two boxes in s have the same color and the same applies to the boxes in t , $C(s_h) \cap C(s_l) = C(t_h) \cap C(t_l) = \emptyset$ also holds. Thus, we have

$$\begin{aligned} \sum_{s \in D_Q, t \in U_Q} |C(s) \cap C(t)| &= \sum_{s \in D_Q, t \in U_Q} (|C(s_h) \cap C(t_h)| + |C(s_h) \cap C(t_l)| + |C(s_l) \cap C(t)|) \\ &= \sum_{s \in D_Q, t \in U_Q} |C(s_h) \cap C(t_h)| + \sum_{s \in D_Q, t \in U_Q} |C(s_h) \cap C(t_l)| + |(\cup_{s \in D_Q} C(s_l)) \cap \cup_{t \in U_Q} C(t)| \end{aligned} \quad (1)$$

For the first term in the last line of Equation 1, we can retrieve $|C(s_h) \cap C(t_h)|$ from the matrix $M(v)$ for each possible pair of s_h and t_h and sum them up. Therefore, the first term can be computed in $O(|D_Q| \cdot |U_Q|)$ time. For the third term, observe that, $\cup_{t \in U_Q} C(t) = C_Q(u_r)$.

Therefore, the third term can be computed by performing, for each color $c \in \cup_{s \in D_Q} C(s_l)$, a 2D orthogonal colored range emptiness query over $P(u_r)$ with c as the query color and Q as the query range. Note that the range emptiness query data structure $E(v_r)$ defined in Section 3.1 is built upon the points $\hat{P}(v_r)$ in rank space. We need to reduce Q into rank space with respect to $\hat{P}(v_r)$ before performing colored range emptiness queries. Since all these queries share the same query range, we need only convert Q into rank space once. This can be done by performing binary searches in $P(v)$ and $P_y(v)$ in $O(\lg n)$ time. As $|\cup_{s \in D_Q} C(s_l)| = |\cup_{s \in D_Q} s_l| = O(|D_Q| \cdot X)$, it requires $O(X|D_Q| \cdot (g(n) + \tau(n)) + \lg n)$ time to compute these colors and then answer all these queries, where $g(n)$ denotes the query time of each range emptiness query in Lemma 1 and $\tau(n)$ denotes the query time of reporting a box and its color in the query range.

Finally, to compute the second term in the last line of Equation 1, observe that,

$$\sum_{s \in D_Q, t \in U_Q} |C(s_h) \cap C(t_l)| = |(\cup_{s \in D_Q} C(s)) \cap (\cup_{t \in U_Q} C(t_l))| - |(\cup_{s \in D_Q} C(s_l)) \cap (\cup_{t \in U_Q} C(t_l))| \quad (2)$$

The first term of the right hand side of Equation 2 can be computed in $O(X|U_Q| \cdot (g(n) + \tau(n)) + \lg n)$ time, again by performing range emptiness queries, but this time we use $E(v_l)$. The second term can be computed by retrieving and sorting the colors in $\cup_{s \in D_Q} C(s_l)$ and those in $\cup_{t \in U_Q} C(t_l)$, and then scanning both sorted lists to compute their intersection. Since $|\cup_{s \in D_Q} C(s_l)|$ (resp. $|\cup_{t \in U_Q} C(t_l)|$) are bounded by $O(X|D_Q|)$ (resp. $O(X|U_Q|)$), the two sets of colors can be retrieved in $O(X|D_Q|\tau(n))$ and $O(X|U_Q|\tau(n))$ time and then sorted using Han's sorting algorithm [15] in $O(X|D_Q|\lg \lg n)$ and $O(X|U_Q|\lg \lg n)$ time. Thus the second term in the last line of Equation 1 can be computed in $O(X(|U_Q| + |D_Q|)(\lg \lg n + \tau(n)))$ time. Overall, computing $|C_Q(u_l) \cap C_Q(u_r)|$ requires $O(|D_Q| \cdot |U_Q| + X(|U_Q| + |D_Q|)(\lg \lg n + g(n) + \tau(n)) + \lg n)$ time. Lemma 4 summarizes the complexities of our framework.

► **Lemma 4.** *Suppose that the 3D stabbing query structure of $S(v_l)$ (or $S(v_r)$) for each node $v \in T$ has duplication factor $\delta(n)$, occupies $O(|P(v)|h(n))$ words, and, given a query point q , it can compute $\phi(n)$ disjoint sets of boxes whose union is the set of boxes containing q in $O(\phi(n))$ time. Furthermore, each subset is a nonempty prefix of a bottom list, and after this prefix is located, its length can be computed in $O(1)$ time and each box in it can be reported in $O(\tau(n))$ time. Let $f(n)$ and $g(n)$ be the functions set in Lemma 1 to implement $E(v_l)$ and $E(v_r)$. Then the structures in our framework occupy $O((n\delta(n)/X)^2 + n \lg n(f(n) + h(n)))$ words and answer a 2D orthogonal colored range counting query in $O(\phi^2(n) + X\phi(n)(\lg \lg n + g(n) + \tau(n)) + \lg n)$ time, where X is an integer parameter in $[1, n]$.*

Proof. Each node v of T stores a pair of lists, $P(v)$ and $P_y(v)$, of points, the colored range emptiness query structures $E(v_l)$ and $E(v_r)$, the stabbing query data structures $S(v_l)$ and $S(v_r)$, and the matrix $M(v)$. Among them, both $P(v)$ and $P_y(v)$ use $|P(v)|$ words of space. $E(v_l)$ and $E(v_r)$ use $O(|P(v)|f(n))$ words of space by Lemma 1. $S(v_l)$ and $S(v_r)$ use $O(|P(v)|h(n))$ words of space. The matrix $M(v)$ uses $O((\delta(n)|P(v)|/X)^2)$ words of space. Summing the space costs over all internal nodes of the range tree T , the overall space cost is at most $\sum_{v \in T} O((\delta(n)|P(v)|/X)^2 + |P(v)|(f(n) + h(n)))$. To simplify this expression, first observe that $\sum_{v \in T} |P(v)| = O(n \lg n)$. Furthermore, we can calculate $\sum_{v \in T} |P(v)|^2$ as follows: At the i -th level of T , there are 2^i nodes, and each node stores a point list of length $n/2^i$. Therefore, the sum of the squares of the lengths of the point lists at the i th level is $n^2/2^i$. Summing up over all levels, we have $\sum_{v \in T} |P(v)|^2 = O(n^2)$. Therefore, the overall space cost simplifies to $O((n\delta(n)/X)^2 + n \lg n(f(n) + h(n)))$.

Given a query range, we use $O(\phi(n))$ time to compute $|C_Q(u_l)|$ and $|C_Q(u_r)|$. As shown before, computing $|C_Q(u_l) \cap C_Q(u_r)|$ can be reduced to computing $\sum_{s \in D_Q, t \in U_Q} |C(s) \cap C(t)|$, which requires $O(|D_Q| \cdot |U_Q| + X(|U_Q| + |D_Q|)(\lg \lg n + g(n) + \tau(n)) + \lg n)$ time. Since both $|D_Q|$ and $|U_Q|$ are upper bounded by $\phi(n)$, the overall query time is $O(\phi^2(n) + X\phi(n)(\lg \lg n + g(n) + \tau(n)) + \lg n)$. \blacktriangleleft

We can now achieve a new time-space tradeoff by using the stabbing queries data structure from Lemma 2 in our framework. To combine Lemmas 2 and 4, observe that $h(n) = O(\lg^2 n)$, $\phi(n) = O(\lg^2 n)$ and $\tau(n) = O(1)$. As discussed before, $\delta(n) = O(\lg^2 n)$. We use part b) of Lemma 1 to implement $E(v_l)$ and $E(v_r)$, so $f(n) = O(\lg \lg n)$ and $g(n) = O(\lg \lg n)$. Hence:

► **Theorem 5.** *Given n colored points on the plane, there is a data structure of $O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$ words of space that answers colored orthogonal range counting queries in $O(\lg^4 n + X \lg^2 n \lg \lg n)$ time, where X is an integer parameter in $[1, n]$. In particular, setting $X = \sqrt{n \lg n}$ yields an $O(n \lg^3 n)$ -word structure with $O(\sqrt{n} \lg^{5/2} n \cdot \lg \lg n)$ query time.*

Unlike our result in Theorem 5, the solution of Kaplan et al. [19] with $O((\frac{n}{X})^2 \lg^6 n + n \lg^4 n)$ words of space and $O(X \lg^7 n)$ query time works under the pointer machine model. Nevertheless, with some modifications, our solution can also be made to work under this same model. First, Lemma 1 requires the word RAM model, we can replace it by the optimal solution to the 2D orthogonal range emptiness query problem by Chazelle [7] with $O(n \lg n / \lg \lg n)$ words of space and $O(\lg n)$ query time. Thus, $g(n) = O(\lg n)$, but the overall space cost of the data structure remains unchanged. Second, when computing $|(\cup_{s \in D_Q} C(s_l)) \cap (\cup_{t \in U_Q} C(t_l))|$, we cannot use Han's sorting algorithm [15] which requires the word RAM. Instead, using mergesort, we can compute this value in $O(X(|U_Q| + |D_Q|) \cdot \lg n)$ time. Finally, to simulate a matrix $M(v)$, we can use lists indexed by binary search trees, so that we can retrieve each entry in $O(\lg n)$ time. Thus, we achieve the following result:

► **Corollary 6.** *Under the arithmetic pointer machine model, given n colored points on the plane, there is a data structure of $O((\frac{n}{X})^2 \lg^4 n + n \lg^3 n)$ words of space that answers colored orthogonal range counting queries in $O(\lg^5 n + X \lg^3 n)$ time, where X is an integer parameter in $[1, n]$. In particular, setting $X = \sqrt{n \lg n}$ yields an $O(n \lg^3 n)$ -word structure with $O(\sqrt{n} \lg^{7/2} n)$ query time.*

4 Two More Solutions with Better Space Efficiency

As the space cost in Theorem 5 is at least $\Omega(n \lg^3 n)$, we now design two more solutions with potentially better space efficiency for 2D orthogonal colored range counting.

4.1 Achieving $O(n \lg^2 n)$ Space

We design an alternative solution for 3D stabbing queries over canonical boxes whose space cost is a logarithmic factor less of that in Lemma 2 asymptotically, and it also satisfies the conditions described in Section 3.1 and can thus be applied in our framework. This leads to another time-space tradeoff for 2D orthogonal colored range counting, whose space cost can be as little as $O(n \lg^2 n)$ by choosing the right parameter value.

This new 3D stabbing query solution requires us to design a data structure supporting 2D dominance counting and reporting, by augmenting a binary range tree constructed over the y -coordinates of the input points. Each internal node v of T is conceptually associated with a list, $P(v)$, of points that are leaf descendants of v , sorted by x -coordinate, but we do not store $P(v)$ explicitly. Lemma 7 presents this data structure. Even though better solutions exist for these problems [17, 4], Lemma 7 gives us additional range tree functionalities that are required for our next two solutions to colored range counting.

► **Lemma 7.** *Consider a binary range tree T constructed over a set, P , of n points on the plane as described above. T can be augmented using $O(n)$ additional words such that, given a query range Q which is the region dominated by a point q , a set, S , of $O(\lg n)$ nodes of T can be located in $O(\lg n)$ time that satisfies the following conditions: For each node $v \in S$, there exists a nonempty prefix $L(v)$ of $P(v)$ such that the point set $P \cap Q$ can be partitioned into $|S|$ disjoint subsets, each consisting of the points in such a prefix. Furthermore, the individual sizes of all these subsets can be computed in $O(\lg n)$ time in total, and each point in such a subset can be reported in $O(\lg^\epsilon n)$ additional time for any positive constant ϵ .*

Proof. For simplicity, we assume that point coordinates are in rank space. Then each leaf of the range tree T represents an integer range $[p.y, p.y]$ if p is stored at this leaf. The range represented by an internal node of T is the union of the ranges represented by its children.

At each internal node v of T , we store a bit vector, $\mathcal{B}(v)$, such that if the point $P(v)[i]$ is a leaf descendant of the left child of v , then $\mathcal{B}(v)[i]$ is set to 0; otherwise $\mathcal{B}(v)[i]$ is set to 1. We construct a data structure of $O(|\mathcal{B}(v)|)$ bits of space upon $\mathcal{B}(v)$ to support the computation of $\text{rank}(v, k)$, which is $\sum_{i \leq k} \mathcal{B}(v)[i]$, for any k in constant time [9]. These bit vectors over all internal nodes v of T use $\sum_v O(|\mathcal{B}(v)|) = O(n \lg n)$ bits, which is $O(n)$ words of space.

Given a query range $Q = [1, q.x] \times [1, q.y]$, we find the path, π , from the root node of T to the $q.y$ -th leaf. For each node u in π , if it is the right child of its parent, we add its left sibling, v , into a set S' . We also add the $q.y$ -th leaf into S' . Then the ranges represented by the nodes in S' form a partition of the query y -range $[1, q.y]$, so $P \cap Q \subseteq \cup_{v \in S'} P(v)$. Furthermore, for each node $v \in S'$, we add it into S if $|P(v) \cap Q| > 0$. To compute $|P(v) \cap Q|$, observe that, since the y -coordinates of points in $P(v)$ are within the query y -range, and these points are increasingly sorted by their x -coordinates, $|P(v) \cap Q|$ is equal to the index, i , of the rightmost point of $P(v)$ whose x -coordinate is no more than $q.x$. Furthermore, if $|P(v) \cap Q| > 0$, then $L(v) = P(v)[1..i]$. Thus the following observation is crucial: Let s and t be two nodes of T , where s is the parent of t , let j be the number of points in $P(s)$ whose x -coordinates are no more than $q.x$. Then, if t is the left child of s , the number of points in $P(t)$ whose x -coordinates are no more than $q.x$ is $|P(s)| - \text{rank}(s, j)$. Otherwise, it is $\text{rank}(s, j)$. We know that at the root node r , the number of points in $P(r)$ whose x -coordinates are no more than $q.x$ is simply $q.x$. Then, during the top down traversal of π , we can make use of this observation and perform up to two **rank** operations at each level, so that for any node $u \in \pi \cup S'$, we can compute the index of the rightmost point of $P(u)$ whose x -coordinate is no more than $q.x$. This way we can compute S and for each $v \in S$, compute $|L(v)|$. The total running time is $O(\lg n)$. To report the coordinates of each point in $L(v)$ for any $v \in S$, we augment T with the *ball inheritance* data structure [4], which can use $O(n)$ additional words of space to support each query in $O(\lg^\epsilon n)$ time. The details are deferred to the full version of the paper. ◀

Next we present the new stabbing query data structure. This time we construct data structures consisting of three layers of trees to answer stabbing queries over a given set of n canonical boxes in three-dimensional space, but a different tree structure is adopted in each layer. At the top-layer we construct a segment tree over the z -coordinates of the boxes. More precisely, we project each box onto the z -axis to obtain an interval, and the segment tree is constructed over all these intervals. A box is assigned to a node in this tree if its corresponding interval on the z -axis is associated with this node. For each node v in the top-layer segment tree, we further construct an interval tree over the projections of the boxes assigned to v on the y -axis, and the interval trees constructed for all the nodes of the top-layer tree form the middle-layer structure. For each node v' of an interval tree, we use

the set $B(v')$, of boxes assigned to it to define the two point sets, $S_{lower}(v')$ and $S_{upper}(v')$, on the plane as follows: We project all the boxes in $B(v')$ onto the xy -plane and get a set of right-open rectangles. Then, $S_{lower}(v')$ is the set of the lower left vertices of these rectangles (henceforth called *lower points*), i.e., $\{(B.x_1, B.y_1) | B \in B(v')\}$, and $S_{upper}(v')$ is the set of the upper left vertices (henceforth called *upper points*), i.e., $\{(B.x_1, B.y_2) | B \in B(v')\}$. We then use Lemma 7 to build a pair of binary range trees, $T_{lower}(v')$ and $T_{upper}(v')$, over $S_{lower}(v')$ and $S_{upper}(v')$, respectively. The range trees constructed for all these interval tree nodes form the bottom-layer structure.

Recall that, each node v'' of a binary range tree in the bottom layer is conceptually associated with a list, $P(v'')$, of lower or upper points, which are the points stored in the leaf descendants of v'' , sorted by x -coordinate. Each point of $P(v'')$ represents a box. Since there is a one-to-one correspondence between a point in $P(v'')$ and the box it represents, we may abuse notation and use $P(v'')$ to refer to the list of boxes that these points represent when the context is clear. Hence $P(v'')$ is the bottom list when the data structure is used in our framework. Lemma 8 summarizes the solution.

► **Lemma 8.** *Given a set of n canonical boxes in three dimension, the data structure above occupies $O(n \lg n)$ words and answers stabbing counting queries in $O(\lg^3 n)$ time and stabbing reporting queries in $O(\lg^3 n + k \cdot \lg^\epsilon n)$ time, where k denotes the number of boxes reported. Furthermore, the set of reported boxes is the union of $O(\lg^3 n)$ different disjoint sets, each of which is a nonempty prefix of some bottom list in the data structure.*

Proof. The top-layer segment tree occupies $O(n \lg n)$ words, while both interval trees and binary ranges trees from Lemma 7 are linear-space data structures. Therefore, the overall space cost is $O(n \lg n)$ words.

To show how to answer a query, let q be the query point. Our query algorithm first searches for $q.z$ in the top-layer segment tree. This locates $O(\lg n)$ nodes of the top-layer tree. Each node v located in this phase stores a list, $B(v)$, of boxes whose z -ranges contain $q.z$. It now suffices to show how to count and report the boxes in each list $B(v)$ whose projections on the xy -plane contain $(q.x, q.y)$. To do this, we use the interval tree in the middle layer that is constructed over $B(v)$. In an interval tree, each node v' stores the median, denoted by $m(v')$, of the endpoints of the intervals associated with its descendants (including itself), and it also stores a set, $I(v')$, of the intervals that contain $m(v')$. In our case, each interval in $I(v')$ is the y -range of a canonical box in $B(v)$, its left endpoint corresponds to the y -coordinate of a point in $S_{lower}(v')$, and its right endpoint corresponds to the y -coordinate of a point in $S_{upper}(v')$. The next phase of our algorithm then starts from the root, r , of this interval tree. If $q.y \leq m(r)$, then an interval in $I(v')$ contains $q.y$ iff its left endpoint is less than or equal to $q.y$. This means, among the points in $S_{lower}(v')$, those lie in $(-\infty, q.x] \times (-\infty, q.y]$ correspond to the boxes whose projections on the xy -plane contain $(q.x, q.y)$. Since the z -range of these boxes already contains $q.z$, they contain q in the three-dimensional space. Hence, by performing a dominance query over $T_{lower}(v')$ using $(-\infty, q.x] \times (-\infty, q.y]$ as the query range, we can compute these boxes. Then, since the nodes in the right subtree r store intervals whose left endpoints are greater than $m(r)$ which is at least $q.y$, none of these intervals can possibly contain $q.y$. Thus, we descend to the left child of r afterwards and repeat this process. If $q.y > m(r)$ instead, then we perform a dominance query over $T_{upper}(v')$ using $[-\infty, q.x] \times (q.y, +\infty)$ as the query range to compute the boxes associated with r that contain q , descend to the right child of r , and repeat.

To analyze the running time, observe that this algorithm locates $O(\lg n)$ nodes in the top-layer segment tree, and for each of these nodes, it further locates $O(\lg n)$ nodes in the middle-layer interval trees. Hence, we perform a 2D dominance counting and reporting query using Lemma 7 for each of these $O(\lg^2 n)$ interval tree nodes, and the proof completes. ◀

In an interval tree, each interval is stored in exactly one node, while in a segment tree or a binary range tree, each interval or point can be associated with $O(\lg n)$ nodes. Therefore, this data structure has duplication factor $\delta = O(\lg^2 n)$. If we combine Lemmas 2 and 4, we also have $h(n) = O(\lg n)$, $\phi(n) = O(\lg^3 n)$ and $\tau(n) = O(\lg^\epsilon n)$. We again use part b) of Lemma 1 to implement $E(v_l)$ and $E(v_r)$, so $f(n) = O(\lg \lg n)$ and $g(n) = O(\lg \lg n)$. Hence:

► **Theorem 9.** *Given n colored points on the plane, there is a data structure of $O((\frac{n}{X})^2 \lg^4 n + n \lg^2 n)$ words of space that answers colored orthogonal range counting queries in $O(\lg^6 n + X \lg^{3+\epsilon} n)$ time, where X is an integer parameter in $[1, n]$ and ϵ is an arbitrary positive constant. In particular, setting $X = \sqrt{n} \lg n$ yields an $O(n \lg^2 n)$ -word structure with $O(\sqrt{n} \lg^{4+\epsilon} n)$ query time.*

4.2 Achieving $O(n \lg n)$ Space

We further improve the space cost of the data structure for 3D stabbing queries over canonical boxes. The new solution also satisfies the conditions described in Section 3.1 and can thus be applied in our framework. This leads to our third time-space tradeoff for 2D orthogonal colored range counting, whose space cost can be as little as $O(n \lg n)$ by choosing the right parameter value.

Our solution will use a space-efficient data structure for 3D orthogonal range searching. Using a range tree with node degree $\lambda \in [2, n]$, we can transform a 2D linear space data structure shown in Lemma 7 into a three-dimensional data structure that uses $O(n \log_\lambda n)$ words. Therefore, we can solve 3D dominance range searching as shown in Lemma 10, whose proof is deferred to the full version of the paper.

► **Lemma 10.** *Given n points in three dimension, there is a data structure of $O(n \log_\lambda n)$ words of space that answers 3D dominance counting queries in $O(\lambda \lg n \cdot \log_\lambda n)$ time and 3D dominance reporting queries in $O(\lambda \lg n \cdot \log_\lambda n + k \lg^\epsilon n)$ time, where k is the number of reported points and λ is an integer parameter in $[2, n]$.*

Next, we design a new data structure for stabbing queries over 3D canonical boxes. It again contains trees of three layers, with interval trees in the top- and middle- layers plus the data structures for 3D dominance range searching from Lemma 10 in the bottom layer. More precisely, the structure at the top layer is an interval tree, T_1 , constructed over the z -coordinates of the boxes. That is, we project each box onto the z -axis to obtain an interval, and the interval tree is constructed over all these intervals. A box is assigned to a node in this tree if its corresponding interval on the z -axis is associated with this node. For each node v in the top-layer interval tree, we further construct an interval tree, $T_2(v)$, over the projections of the boxes assigned to v on the y -axis, and the interval trees constructed for all the nodes of the top-layer tree form the middle-layer structure. For each node v' of an interval tree in the middle-layer, we use the set $B(v')$, of boxes assigned to it to define the four point sets, $S_{ll}(v')$, $S_{lr}(v')$, $S_{ul}(v')$, and $S_{ur}(v')$ in 3D such that $S_{ll}(v') = \{(B.x_1, B.y_1, B.z_1) | B \in B(v')\}$, $S_{lr}(v') = \{(B.x_1, B.y_1, B.z_2) | B \in B(v')\}$, $S_{ul}(v') = \{(B.x_1, B.y_2, B.z_1) | B \in B(v')\}$, and $S_{ur}(v') = \{(B.x_1, B.y_2, B.z_2) | B \in B(v')\}$. We then use Lemma 10 to build a set of 3D dominance range searching structures, $T_{ll}(v')$, $T_{lr}(v')$, $T_{ul}(v')$ and $T_{ur}(v')$, over $S_{ll}(v')$, $S_{lr}(v')$, $S_{ul}(v')$, and $S_{ur}(v')$, respectively. The 3D dominance range searching structures constructed for the nodes of all interval trees in the middle-layer form the bottom-layer structure.

As shown in the previous section, we need to identify the bottom lists from the data structures described above, which would be used in our framework. Without loss of generality, we take the 3D dominance range searching structure, $T_{ll}(v')$, built in the bottom-layer as

an example. Observe that $T_{ll}(v')$ is a λ -ary range tree, of which each internal node, v'' , stores a binary range tree data structure $T_b(v'')$ implemented by Lemma 7 for 2D dominance range searching. Recall that, each node \hat{v} of a binary range tree $T_b(v'')$ is conceptually associated with a list, $P(\hat{v})$, of points from the set $S_{ll}(v')$, which are the points stored in the leaf descendants of \hat{v} , sorted by x -coordinate. Each point of $P(\hat{v})$ represents a box. Since there is a one-to-one correspondence between a point in $P(\hat{v})$ and the box it represents, we may abuse notation and use $P(\hat{v})$ to refer to the list of boxes that these points represent when the context is clear. Hence $P(\hat{v})$ is the bottom list when the data structure is used in our framework. The following lemma summarizes this solution:

► **Lemma 11.** *Given a set of n canonical boxes in three dimension, the data structure described above occupies $O(n \log_\lambda n)$ words of space and answers stabbing counting queries in $O(\lg^2 n \cdot \lambda \lg n \cdot \log_\lambda n)$ time and stabbing reporting queries in $O(\lg^2 n \cdot \lambda \lg n \cdot \log_\lambda n + k \cdot \lg^\epsilon n)$ time, where k denotes the number of boxes reported and λ is an integer parameter in $[2, n]$. Furthermore, the set of reported boxes is the union of $O(\lg^2 n \cdot \lambda \lg n \cdot \log_\lambda n)$ different disjoint sets, each of which is a nonempty prefix of some bottom list in the data structure.*

Proof. The top- and middle- layer interval trees are linear-space data structure, while the 3D dominance range searching structures from Lemma 10 occupy $O(n \log_\lambda n)$ words of space in total. Therefore, the overall space cost is $O(n \log_\lambda n)$ words.

To show how to answer a query, let $q = (q.x, q.y, q.z)$ be the query point. In an interval tree T_1 (resp. $T_2(v)$), each node v (resp. v') stores the median, denoted by $m_z(v)$ (resp. $m_y(v')$), of the endpoints of the intervals associated with its descendants (including itself), and it also stores a set, $I_z(v)$ (resp. $I_y(v')$), of the intervals that contain $m_z(v)$ (resp. $m_y(v')$). In our case, each interval in $I_z(v)$ (resp. $I_y(v')$) is the z -range (resp. y -range) of a canonical box in $B(v)$ (resp. $B(v')$). The query algorithm starts from the root, r , of T_1 . If $q.z \leq m_z(r)$, then an interval in $I_z(r)$ contains $q.z$ iff its left endpoint is less than or equal to $q.z$. Then we visit the interval tree $T_2(r)$ in the middle-layer that is constructed upon the y -ranges of the boxes in $B(r)$. The next phase of our algorithm then starts from the root, r' , of $T_2(r)$. If $q.y \leq m_y(r')$, then an interval in $I_y(r')$ contains $q.y$ iff its lower endpoint is less than or equal to $q.y$. This means, among the points in $S_{ll}(v')$, those lie in $(-\infty, q.x] \times (-\infty, q.y] \times (-\infty, q.z]$ correspond to the boxes containing $(q.x, q.y, q.z)$. Hence, by performing a dominance query over $T_{ll}(v')$ using $(-\infty, q.x] \times (-\infty, q.y] \times (-\infty, q.z]$ as the query range, we can compute these boxes. Then, since the nodes in the right subtree r' store intervals whose lower endpoints are greater than $m_y(r')$ which is at least $q.y$, none of these intervals can possibly contain $q.y$. Thus, we descend to the left child of r' afterwards and repeat this process. Otherwise if $q.y > m_y(r')$ instead, then we perform a dominance query over $T_{ul}(v')$ using $(-\infty, q.x] \times (q.y, +\infty) \times (-\infty, q.z]$ as the query range to compute the boxes associated with r' that contain q , descend to the right child of r' , and repeat this process until reaching the leaf level of $T_2(r)$. Once $T_2(r)$ has been traversed, we return the root node r of T_1 . Since the nodes in the right subtree r store intervals whose left endpoints are greater than $m_z(r)$ which is at least $q.z$, none of these intervals can possibly contain $q.z$. Thus, we descend to the left child of r afterwards and repeat this process. Otherwise, if $q.z > m_z(r)$ instead, then we perform a dominance query over either $T_{lr}(v')$ using $(-\infty, q.x] \times (-\infty, q.y] \times (q.z, +\infty)$ as the query range or $T_{ur}(v')$ using $[-\infty, q.x] \times (q.y, +\infty) \times (q.z, +\infty)$ as the query range to compute the boxes associated with v' that contain q , where v' is a node of $T_2(r')$ that we visit, descend to the right child of r , and repeat the process. In summary, let v denote a node on the traversed path of T_1 and given node v , let v' denote a node on the traversed path of $T_2(v)$. By comparing $q.z$ against $m_z(v)$ and $q.y$ against $m_y(v')$, we can decide which

3D dominance range searching data structure to be used in the bottom-layer. It includes the following four different cases:

- when $q.z \leq m_z(v)$ and $q.y \leq m_y(v')$, we search $T_{ll}(v')$ for the points of $S_{ll}(v')$ in the query range $(-\infty, q.x] \times (-\infty, q.y] \times (-\infty, q.z]$;
- when $q.z \leq m_z(v)$ and $q.y > m_y(v')$, we search $T_{ul}(v')$ for the points of $S_{ul}(v')$ in the query range $(-\infty, q.x] \times (q.y, +\infty) \times (-\infty, q.z]$;
- when $q.z > m_z(v)$ and $q.y \leq m_y(v')$, we search $T_{lr}(v')$ for the points of $S_{lr}(v')$ in the query range $(-\infty, q.x] \times (-\infty, q.y] \times (q.z, +\infty)$;
- when $q.z > m_z(v)$ and $q.y > m_y(v')$, we search $T_{ur}(v')$ for the points of $S_{ur}(v')$ in the query range $(-\infty, q.x] \times (q.y, +\infty) \times (q.z, +\infty)$.

To analyze the running time, observe that this algorithm locates $O(\lg n)$ nodes in the top-layer interval tree, and for each of these nodes, it further locates $O(\lg n)$ nodes in the middle-layer interval trees. Hence, we perform a 3D dominance counting and reporting query using Lemma 10 for each of these $O(\lg^2 n)$ interval tree nodes, and the proof completes. ◀

In an interval tree, each interval is stored in exactly one node, while in a 3D dominance range searching structure from Lemma 10, each point can be associated with $O(\lg n \cdot \log_\lambda n)$ nodes. Therefore, this data structure has duplication factor $\delta(n) = O(\lg n \cdot \log_\lambda n)$. If we combine Lemmas 11 and 4, we have $h(n) = O(\log_\lambda n)$, $\phi(n) = O(\lg^2 n \cdot \lambda \lg n \cdot \log_\lambda n)$ and $\tau(n) = O(\lg^\epsilon n)$. We use part a) of Lemma 1 to implement $E(v_l)$ and $E(v_r)$, so $f(n) = O(1)$ and $g(n) = O(\lg^\epsilon n)$. Hence:

► **Theorem 12.** *Given n colored points on the plane, there is a data structure of $O((\frac{n}{X})^2 \lg^2 n \cdot \log_\lambda^2 n + n \lg n \cdot \log_\lambda n)$ words of space that answers colored orthogonal range counting queries in $O(\lambda^2 \cdot \lg^6 n \cdot \log_\lambda^2 n + X \cdot \lg^{3+\epsilon} n \cdot \lambda \log_\lambda n)$ time, where X is an integer parameter in $[1, n]$, λ is an integer parameter in $[2, n]$, and ϵ is any constant in $(0, 1)$. Setting $X = \sqrt{n \lg n \log_\lambda n}$ and $\lambda = \lg^\epsilon n$ yields an $O(n \frac{\lg^2 n}{\lg \lg n})$ -word structure with $O(\sqrt{n} \lg^{5+\epsilon'} n)$ query time for any constant $\epsilon' > 2\epsilon$. Alternatively, setting $X = \sqrt{n \lg n}$ and $\lambda = n^{\epsilon/5}$ yields an $O(n \lg n)$ -word structure with $O(n^{1/2+\epsilon})$ query time.*

References


- 1 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory Comput.*, 8(1):69–94, 2012.
- 2 Timothy M. Chan. Speeding up the Four Russians Algorithm by About One More Logarithmic Factor. In *SODA*, pages 212–217, 2015.
- 3 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further results on colored range searching. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry, SoCG 2020, June 23–26, 2020, Zürich, Switzerland*, volume 164 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 4 Timothy M Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *27th Symposium on Computational Geometry*, pages 1–10. ACM, 2011.
- 5 Timothy M Chan and Yakov Nekrich. Better data structures for colored orthogonal range reporting. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 627–636. SIAM, 2020.
- 6 Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, pages 268–279, 2000.
- 7 Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

- 8 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.
- 9 David R Clark and J Ian Munro. Efficient suffix trees on secondary storage. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 383–391, 1996.
- 10 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct color searching in one dimension. 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 30:1–30:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 11 Roberto Grossi and Søren Vind. Colored range searching in linear space. In *Scandinavian Workshop on Algorithm Theory*, pages 229–240. Springer, 2014.
- 12 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel HM Smid. Computational geometry: Generalized (or colored) intersection searching. *Handbook of Data Structures and Applications*, pages 1042–1057, 2018.
- 13 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.
- 14 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Algorithms for generalized halfspace range searching and other intersection searching problems. *Computational Geometry*, 6(1):1–19, 1996.
- 15 Yijie Han. Deterministic sorting in $o(n \log \log n)$ time and linear space. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 602–608, 2002.
- 16 Meng He and Serikzhan Kazi. Data structures for categorical path counting queries. In *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*, To appear.
- 17 Joseph JáJá, Christian W Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *International Symposium on Algorithms and Computation*, pages 558–568. Springer, 2004.
- 18 Ravi Janardan and Mario Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(01):39–69, 1993.
- 19 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008.
- 20 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 52–60, 2006.
- 21 Ying Kit Lai, Chung Keung Poon, and Benyun Shi. Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms*, 6(3):420–432, 2008.
- 22 J. Ian Munro, Yakov Nekrich, and Sharma V. Thankachan. Range counting with distinct constraints. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, pages 83–88. Queen’s University, Ontario, Canada, 2015. URL: <http://research.cs.queensu.ca/cccg2015/CCCG15-papers/44.pdf>.
- 23 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems (TODS)*, 39(1):1–21, 2014.
- 24 Saladi Rahul. Approximate range counting revisited. *Journal of Computational Geometry*, 12(1):40–69, 2021.
- 25 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898, 2012.
- 26 Huacheng Yu. An improved combinatorial algorithm for Boolean matrix multiplication. *Inf. Comput.*, 261:240–247, 2018.

Computing the 4-Edge-Connected Components of a Graph in Linear Time

Loukas Georgiadis ✉ 

Department of Computer Science & Engineering, University of Ioannina, Greece

Giuseppe F. Italiano ✉ 

LUISS University, Rome, Italy

Evangelos Kosinas ✉

Department of Computer Science & Engineering, University of Ioannina, Greece

Abstract

We present the first linear-time algorithm that computes the 4-edge-connected components of an undirected graph. Hence, we also obtain the first linear-time algorithm for testing 4-edge connectivity. Our results are based on a linear-time algorithm that computes the 3-edge cuts of a 3-edge-connected graph G , and a linear-time procedure that, given the collection of all 3-edge cuts, partitions the vertices of G into the 4-edge-connected components.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Cuts, Edge Connectivity, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.47

Related Version *Full Version:* <https://arxiv.org/abs/2105.02910> [6]

Funding Research at the University of Ioannina supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant”, Project FANTA (eEfficient Algorithms for NeTwork Analysis), number HFRI-FM17-431. G. F. Italiano is partially supported by MIUR, the Italian Ministry for Education, University and Research, under PRIN Project AHeAD (Efficient Algorithms for HARnessing Networked Data).

1 Introduction

Let $G = (V, E)$ be a connected undirected graph with m edges and n vertices. An (*edge*) *cut* of G is a set of edges $S \subseteq E$ such that $G \setminus S$ is not connected. We say that S is a k -*cut* if its cardinality is $|S| = k$. Also, we refer to the 1-cuts as the *bridges* of G . A cut S is *minimal* if no proper subset of S is a cut of G . The *edge connectivity* of G , denoted by $\lambda(G)$, is the minimum cardinality of an edge cut of G . A graph is k -*edge-connected* if $\lambda(G) \geq k$.

A cut S separates two vertices u and v , if u and v lie in different connected components of $G \setminus S$. Vertices u and v are k -edge-connected, denoted by $u \stackrel{G}{\equiv}_k v$, if there is no $(k-1)$ -cut that separates them. By Menger’s theorem [16], u and v are k -edge-connected if and only if there are k -edge-disjoint paths between u and v . A k -*edge-connected component* of G is a maximal set $C \subseteq V$ such that there is no $(k-1)$ -edge cut in G that disconnects any two vertices $u, v \in C$ (i.e., u and v are in the same connected component of $G \setminus S$ for any $(k-1)$ -edge cut S). We can define, analogously, the *vertex cuts* and the k -*vertex-connected components* of G .

Computing and testing the edge connectivity of a graph, as well as its k -edge-connected components, is a classical subject in graph theory, as it is an important notion in several application areas (see, e.g., [19]), that has been extensively studied since the 1970’s. It is known how to compute the $(k-1)$ -edge cuts, $(k-1)$ -vertex cuts, k -edge-connected components



© Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 47; pp. 47:1–47:17

Leibniz International Proceedings in Informatics



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

and k -vertex-connected components of a graph in linear time for $k \in \{2, 3\}$ [5, 10, 18, 21, 25]. The case $k = 4$ has also received significant attention [2, 3, 11, 12]. Unfortunately, none of the previous algorithms achieved linear running time. In particular, Kanevsky and Ramachandran [11] showed how to test whether a graph is 4-vertex-connected in $O(n^2)$ time. Furthermore, Kanevsky et al. [12] gave an $O(m + n\alpha(m, n))$ -time algorithm to compute the 4-vertex-connected components of a 3-vertex-connected graph, where α is a functional inverse of Ackermann's function [23]. Using the reduction of Galil and Italiano [5] from edge connectivity to vertex connectivity, the same bounds can be obtained for 4-edge connectivity. Specifically, one can test whether a graph is 4-edge-connected in $O(n^2)$ time, and one can compute the 4-edge-connected components of a 3-edge-connected graph in $O(m + n\alpha(m, n))$ time. Dinitz and Westbrook [3] presented an $O(m + n \log n)$ -time algorithm to compute the 4-edge-connected components of a general graph G (i.e., when G is not necessarily 3-edge-connected). Nagamochi and Watanabe [20] gave an $O(m + k^2 n^2)$ -time algorithm to compute the k -edge-connected components of a graph G , for any integer k . We also note that the edge connectivity of a simple undirected graph can be computed in $O(m \text{polylog} n)$ time, randomized [8, 13] or deterministic [9, 15]. The best current bound is $O(m \log^2 n \log \log^2 n)$, achieved by Henzinger et al. [9] which provided an improved version of the algorithm of Kawarabayashi and Thorup [15].

Our results and techniques. In this paper we present the first linear-time algorithm that computes the 4-edge-connected components of a general graph G , thus resolving a problem that remained open for more than 20 years. Hence, this also implies the first linear-time algorithm for testing 4-edge connectivity. We base our results on the following ideas. First, we extend the framework of Georgiadis and Kosinas [7] for computing 2-edge cuts (as well as mixed cuts consisting of a single vertex and a single edge) of G . Similar to known linear-time algorithms for computing 3-vertex-connected and 3-edge-connected components [10, 25], Georgiadis and Kosinas [7] define various concepts with respect to a depth-first search (DFS) spanning tree of G . We extend this framework by introducing new key parameters that can be computed efficiently and provide characterizations of the various types of 3-edge cuts that may appear in a 3-edge-connected graph. We deal with the general case by dividing G into auxiliary graphs H_1, \dots, H_ℓ , such that each H_i is 3-edge-connected and corresponds to a different 3-edge-connected component of G . Also, for any two vertices x and y , we have $x \stackrel{G}{\equiv}_4 y$ if and only if x and y are both in the same auxiliary graph H_i and $x \stackrel{H_i}{\equiv}_4 y$. Furthermore, this reduction allows us to compute in linear time the number of *minimal 3-edge cuts* in a general graph G . Next, in order to compute the 4-edge-connected components in each auxiliary graph H_i , we utilize the fact that a minimum cut of a graph G separates G into two connected components. Hence, we can define the set V_C of the vertices in the connected component of $G \setminus C$ that does not contain a specified root vertex r . We refer to the number of vertices in V_C as the r -size of the cut C . Then, we apply a recursive algorithm that successively splits H_i into smaller graphs according to its 3-cuts. When no more splits are possible, the connected components of the final split graph correspond to the 4-edge-connected components of G . We show that we can implement this procedure in linear time by processing the cuts in non-decreasing order with respect to their r -size.

Due to the space constraints we omit several technical details and proofs. They can be found in the full version of the paper which is available at [6].

2 Concepts defined on a DFS-tree structure

Let $G = (V, E)$ be a connected undirected graph, which may have multiple edges. For a set of vertices $S \subseteq V$, the induced subgraph of S , denoted by $G[S]$, is the subgraph of G with vertex set S and edge set $\{e \in E \mid \text{both ends of } e \text{ lie in } S\}$. Let T be the spanning tree of G provided by a depth-first search (DFS) of G [21], with start vertex r . The edges in T are called tree-edges; the edges in $E \setminus T$ are called back-edges, as their endpoints have ancestor-descendant relation in T . A vertex u is an ancestor of a vertex v (v is a descendant of u) if the tree path from r to v contains u . Thus, we consider a vertex to be an ancestor (and, consequently, a descendant) of itself. We let $p(v)$ denote the parent of a vertex v in T . If u is a descendant of v in T , we denote the set of vertices of the simple tree path from u to v as $T[u, v]$. The expressions $T[u, v)$ and $T(u, v]$ have the obvious meaning (i.e., the vertex on the side of the parenthesis is excluded). From now on, we identify vertices with their preorder number (assigned during the DFS). Thus, v being an ancestor of u in T implies that $v \leq u$. Let $T(v)$ denote the set of descendants of v , and let $ND(v)$ denote the number of descendants of v (i.e. $ND(v) = |T(v)|$). With all $ND(v)$ computed, we can check in constant time whether a vertex u is a descendant of v , since $u \in T(v)$ if and only if $v \leq u$ and $u < v + ND(v)$ [22].

Whenever (x, y) denotes a back-edge, we shall assume that x is a descendant of y . We let $B(v)$ denote the set of back-edges (x, y) , where x is a descendant of v and y is a proper ancestor of v . Thus, if we remove the tree-edge $(v, p(v))$, $T(v)$ remains connected to the rest of the graph through the back-edges in $B(v)$. This implies that G is 2-edge-connected if and only if $|B(v)| > 0$, for every $v \neq r$. Furthermore, G is 3-edge-connected only if $|B(v)| > 1$, for every $v \neq r$. We let $b_count(v)$ denote the number of elements of $B(v)$ (i.e. $b_count(v) = |B(v)|$). $low(v)$ denotes the lowest y such that there exists a back-edge $(x, y) \in B(v)$. Similarly, $high(v)$ is the highest y such that there exists a back-edge $(x, y) \in B(v)$.

We let $M(v)$ denote the nearest common ancestor of all x for which there exists a back-edge $(x, y) \in B(v)$. Note that $M(v)$ is a descendant of v . Let m be a vertex and v_1, \dots, v_k be all the vertices with $M(v_1) = \dots = M(v_k) = m$, sorted in decreasing order. (Observe that v_{i+1} is an ancestor of v_i , for every $i \in \{1, \dots, k-1\}$, since m is a common descendant of all v_1, \dots, v_k .) Then we have $M^{-1}(m) = \{v_1, \dots, v_k\}$, and we define $nextM(v_i) := v_{i+1}$, for every $i \in \{1, \dots, k-1\}$, and $lastM(v_i) := v_k$, for every $i \in \{1, \dots, k\}$. Thus, for every vertex v , $nextM(v)$ is the successor of v in the decreasingly sorted list $M^{-1}(M(v))$, and $lastM(v)$ is the lowest element in $M^{-1}(M(v))$.

The following two facts have been proved in [7].

► **Fact 1.** All $ND(v)$, $b_count(v)$, $M(v)$, $low(v)$ and $high(v)$ can be computed in total linear-time, for all vertices v .

► **Fact 2.** $B(u) = B(v) \Leftrightarrow M(u) = M(v)$, and $high(u) = high(v) \Leftrightarrow M(u) = M(v)$ and $b_count(u) = b_count(v)$.

Furthermore, [7] implies the following characterization of a 3-edge-connected graph.

► **Fact 3.** G is 3-edge-connected if and only if $|B(v)| > 1$, for every $v \neq r$, and $B(v) \neq B(u)$, for every pair of vertices u and v , $u \neq v$.

Now let us provide some extensions of those concepts that will be needed for our purposes. Assume that G is 3-edge-connected, and let $v \neq r$ be a vertex of G . By Fact 3, $b_count(v) > 1$, and therefore there are at least two back-edges in $B(v)$. Thus, there is at least one back-edge $(x, y) \in B(v)$ such that $y = low(v)$. We let $low1(v)$ denote y , and $low1D(v)$ denote x . In other

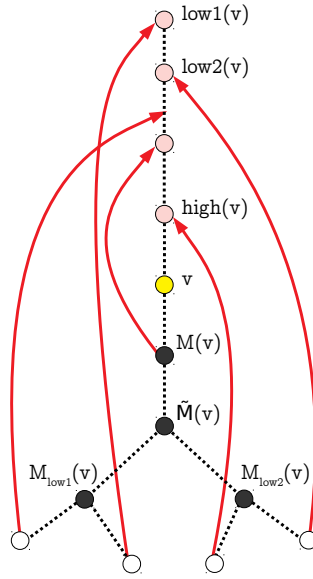
words, $low1(v)$ is the *low* point of v , and $low1D(v)$ is a descendant of v which is connected with a back-edge to its *low* point. (Notice, however, that $low1D(v)$ is not uniquely determined.) Similarly, we let $highD(v)$ denote a descendant of v which is connected with a back-edge to the *high* point of v . Then we define $low2(v) := \min\{y' \mid \exists(x', y') \in B(v) \setminus \{(low1D(v), low1(v))\}\}$, and we let $low2D(v)$ denote a descendant of v which is connected with a back-edge to $low2(v)$. Thus, if $v \neq r$, we have that $(low1D(v), low(v))$ and $(low2D(v), low2(v))$ are two distinct back-edges in $B(v)$. It is easy to compute all $low1(v)$, $low1D(v)$, $low2(v)$ and $low2D(v)$ during the DFS. It is also easy to extend the algorithm for the computation of *high* points in order to compute all $highD(v)$. (We refer to [6] for the details.)

We let $l(v)$ denote the lowest y for which there exists a back-edge (v, y) , or v if no such back-edge exists. Thus, $low(v) \leq l(v)$. Now let c_1, \dots, c_k be the children of v sorted in non-decreasing order w.r.t. their *low* point. Then we call c_1 the *low1* child of v , and c_2 the *low2* child of v . (Of course, the *low1* and *low2* children of v are not uniquely determined after a DFS on G , since we may have $low(c_1) = low(c_2)$.) We let $\tilde{M}(v)$ denote the nearest common ancestor of all x for which there exists a back-edge $(x, y) \in B(v)$ with x a proper descendant of $M(v)$. Formally, $\tilde{M}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \neq M(v)\}$. If the set $\{x \mid \exists(x, y) \in B(v) \text{ and } x \neq M(v)\}$ is empty, we leave $\tilde{M}(v)$ undefined. We also define $M_{low1}(v)$ as the nearest common ancestor of all x for which there exists a back-edge $(x, y) \in B(v)$ with x being a descendant of the *low1* child of $M(v)$, and also define $M_{low2}(v)$ as the nearest common ancestor of all x for which there exists a back-edge $(x, y) \in B(v)$ with x a descendant of the *low2* child of $M(v)$. Formally, $M_{low1}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \text{ is a descendant of the } low1 \text{ child of } M(v)\}$ and $M_{low2}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \text{ is a descendant of the } low2 \text{ child of } M(v)\}$. If the set in the formal definition of $M_{low1}(v)$ (resp. $M_{low2}(v)$) is empty, we leave $M_{low1}(v)$ (resp. $M_{low2}(v)$) undefined.

The following list summarizes the concepts that we use on a DFS-tree; they are defined for all $v \neq r$. (For an illustration, see Figure 1.)

- $B(v) := \{(x, y) \mid x \text{ is a descendant of } v \text{ and } y \text{ is a proper ancestor of } v\}$.
- $l(v) := \min(\{y \mid \exists(v, y) \in B(v)\} \cup \{v\})$.
- $low(v) := \min\{y \mid \exists(x, y) \in B(v)\}$.
- $low1(v) := low(v)$.
- $low1D(v) := \text{a vertex } x \text{ such that } (x, low1(v)) \in B(v)$.
- $low2(v) := \min\{y' \mid \exists(x', y') \in B(v) \setminus \{(low1D(v), low1(v))\}\}$.
- $low2D(v) := \text{a vertex } x \text{ such that } (x, low2(v)) \in B(v)$.
- $high(v) := \max\{y \mid \exists(x, y) \in B(v)\}$.
- $highD(v) := \text{a vertex } x \text{ such that } (x, high(v)) \in B(v)$.
- $M(v) := nca\{x \mid \exists(x, y) \in B(v)\}$.
- $\tilde{M}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \text{ is a proper descendant of } M(v)\}$.
- $M_{low1}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \text{ is a descendant of the } low1 \text{ child of } M(v)\}$.
- $M_{low2}(v) := nca\{x \mid \exists(x, y) \in B(v) \text{ and } x \text{ is a descendant of the } low2 \text{ child of } M(v)\}$.

In Section 2.1 of the full paper [6], we show how to compute all these concepts in linear time.



■ **Figure 1** An illustration of some concepts defined on a DFS-tree. The red arrows correspond to the back-edges in $B(v)$. Dashed lines correspond to tree-paths.

3 Computing the 3-cuts of a 3-edge-connected graph

In this section we present a linear-time algorithm that computes all the 3-edge-cuts of a 3-edge-connected graph $G = (V, E)$. It is well-known that the number of the 3-edge-cuts of G is $O(n)$ [19] (e.g., it follows from the definition of the cactus graph [1, 14]), but we provide an independent proof of this fact. Then, in Section 4.1, we show how to extend this algorithm so that it can also count the number of minimal 3-edge-cuts of a general graph. Note that there can be $O(n^3)$ such cuts [2].

Our method is to classify the 3-cuts on the DFS-tree T in a way that allows us to compute them efficiently. If $\{e_1, e_2, e_3\}$ is a 3-cut, we can initially distinguish three cases w.l.o.g.: either e_1 is a tree-edge and both e_2 and e_3 are back-edges, or e_1 and e_2 are two tree-edges and e_3 is a back-edge, or e_1, e_2 and e_3 is a triplet of tree-edges. Then, we divide those cases in subcases based on the concepts we have introduced in the previous section. Figure 2 gives a general overview of the cases we will handle in some detail in the following sections.

3.1 One tree-edge and two back-edges

The following lemma characterizes all 3-cuts consisting of a tree-edge and two back-edges.

► **Lemma 4.** *Let $\{(u, p(u)), e, e'\}$ be a 3-cut such that e and e' are back-edges. Then $B(u) = \{e, e'\}$. Conversely, if for a vertex $u \neq r$ we have $B(u) = \{e, e'\}$ where e and e' are back-edges, then $\{(u, p(u)), e, e'\}$ is a 3-cut.*

Thus, to compute all the 3-cuts of this type, we have to find all $u \neq r$ with $b_count(u) = 2$. For every such u , there are two back-edges e_1, e_2 such that $B(u) = \{e_1, e_2\}$, and so, w.l.o.g., we have $e_1 = (low1D(u), low1(u))$ and $e_2 = (low2D(u), low2(u))$. Then we mark $\{(u, p(u)), e_1, e_2\}$ as a 3-cut.

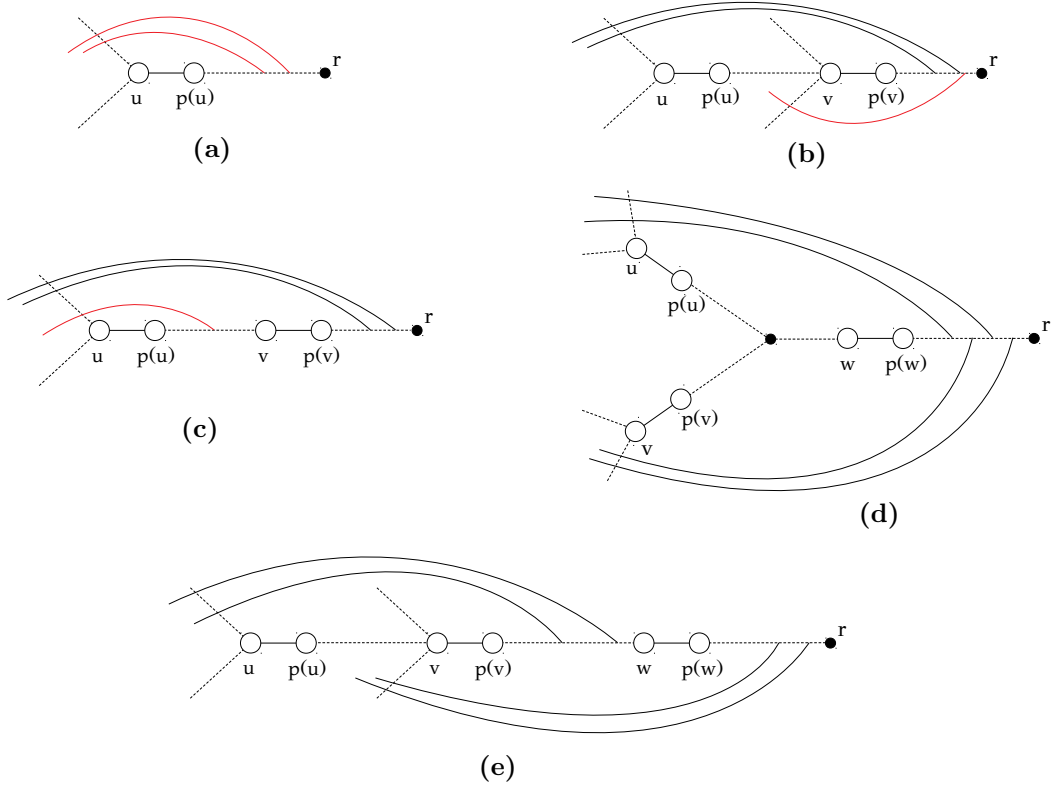


Figure 2 The types of 3-cuts with respect to a DFS-tree. (a) One tree-edge $(u, p(u))$ and two back-edges. (b) Two tree-edges $(u, p(u))$ and $(v, p(v))$, where u is a descendant of v , and one back-edge in $B(v) \setminus B(u)$. (c) Two tree-edges $(u, p(u))$ and $(v, p(v))$, where u is a descendant of v , and one back-edge in $B(u) \setminus B(v)$. (d) Three tree-edges $(u, p(u))$, $(v, p(v))$ and $(w, p(w))$, where w is an ancestor of u and v , but u and v are not related as ancestor and descendant. (e) Three tree-edges $(u, p(u))$, $(v, p(v))$ and $(w, p(w))$, where u is a descendant of v and v is a descendant of w .

3.2 Two tree-edges and one back-edge

In the case of 3-cuts consisting of two tree-edges and a back-edge, we have the following.

► Lemma 5. *Let $\{(u, p(u)), (v, p(v)), e\}$ be a 3-cut such that e is a back-edge. Then u and v are related as ancestor and descendant.*

► Proposition 6. *Let $\{(u, p(u)), (v, p(v)), e\}$ be a 3-cut, where e is a back-edge. Then, either (1) $B(v) = B(u) \sqcup \{e\}$ or (2) $B(u) = B(v) \sqcup \{e\}$. Conversely, if there exists a back-edge e such that (1) or (2) is true, then $\{(u, p(u)), (v, p(v)), e\}$ is a 3-cut.*

We let $V(u)$, for a $u \neq r$, be the set of all v that are ancestors of u such that $B(v) = B(u) \sqcup \{e\}$, for a back-edge e . We also let $U(v)$, for a $v \neq r$, be the set of all u that are descendants of v such that $B(u) = B(v) \sqcup \{e\}$, for a back-edge e . Then, for every 3-cut $\{(u, p(u)), (v, p(v)), e\}$, where e is a back-edge, Proposition 6 implies that either $u \in V(u)$ or $v \in U(v)$.

The following two lemmata imply that the number of 3-cuts consisting of two tree-edges and a back-edge is $O(n)$.

► Lemma 7. *Let v, v' be two distinct vertices. Then $V(u) \cap V(u') = \emptyset$.*

► Lemma 8. *Let u, u' be two distinct vertices. Then $U(v) \cap U(v') = \emptyset$.*

Now, every $v \in V(u)$ has either $\tilde{M}(v) = M(u)$, or $M_{low1}(v) = M(u)$, or $M_{low2}(v) = M(u)$, and u is the lowest vertex which is greater than v such that $\tilde{M}(v) = M(u)$, or $M_{low1}(v) = M(u)$, or $M_{low2}(v) = M(u)$, respectively. This suggests a method to compute, for every vertex v , the unique u (if it exists) such that $v \in V(u)$. We process all vertices v , and for every v that we process we have to find the lowest element u of $M^{-1}(x)$ which is greater than v , for every $x \in \{\tilde{M}(v), M_{low1}(v), M_{low2}(v)\}$, and check whether $v \in V(u)$. To perform this efficiently, we have the lists $M^{-1}(x)$, for every vertex x , sorted in decreasing order, and we process the vertices in a bottom-up fashion. Then, for every v that we process, and every $x \in \{\tilde{M}(v), M_{low1}(v), M_{low2}(v)\}$, we search for the lowest u in $M^{-1}(x)$ which is greater than v , by traversing the list $M^{-1}(x)$ starting from the last element of $M^{-1}(x)$ that we considered, which is being stored in a variable `currentVertex[x]`. This is to avoid traversing $M^{-1}(x)$ from the beginning each time we process a vertex v . We can check in constant time whether $v \in V(u)$ thanks to the following lemma.

► **Lemma 9.** *Let v be an ancestor of u such that either $\tilde{M}(v) = M(u)$, or $M_{low1}(v) = M(u)$, or $M_{low2}(v) = M(u)$, and let $m = \tilde{M}(v)$, or $m = M_{low1}(v)$, or $m = M_{low2}(v)$, depending on whether $\tilde{M}(v) = M(u)$, or $M_{low1}(v) = M(u)$, or $M_{low2}(v) = M(u)$, respectively. Then, $v \in V(u)$ if and only if u is the lowest element in $M^{-1}(m)$ which is greater than v and such that $high(u) < v$ and $b_count(v) = b_count(u) + 1$.*

Finally, for a $v \in V(u)$, we can immediately identify the back-edge (x, y) with $B(v) = B(u) \sqcup \{(x, y)\}$, since we have $x = \tilde{M}(v)$ and $y = l(\tilde{M}(v))$, or $x = M_{low1}(v)$ and $y = l(M_{low1}(v))$, or $x = M_{low2}(v)$ and $y = l(M_{low2}(v))$, depending on whether $\tilde{M}(v) = M(u)$, or $M_{low1}(v) = M(u)$, or $M_{low2}(v) = M(u)$, respectively. Algorithm 1 shows how we can compute all 3-cuts of the form $\{(u, p(u)), (v, p(v)), e\}$, with $B(v) = B(u) \sqcup \{e\}$.

We can use a similar method to compute the 3-cuts of the form $\{(u, p(u)), (v, p(v)), e\}$, with $B(u) = B(v) \sqcup \{e\}$.

3.3 Three tree-edges

The case of 3-cuts consisting of three tree-edges is more involved and is subdivided into several subcases. The following is generally true for all such 3-cuts.

► **Lemma 10.** *Let $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ be a 3-cut, and assume, without loss of generality, that $w < \min\{v, u\}$. Then w is an ancestor of both u and v .*

First we treat the case that u and v are not related as ancestor and descendant. We have the following characterizations of the 3-cuts of this type.

► **Proposition 11.** *Let u and v be two vertices which are not related as ancestor and descendant, and let w be an ancestor of both u and v . Then $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut if and only if $B(w) = B(u) \sqcup B(v)$.*

► **Lemma 12.** *Let u and v be two vertices which are not related as ancestor and descendant, and let w be an ancestor of both u and v . Then $B(w) = B(u) \sqcup B(v)$ if and only if: $M_{low1}(w) = M(u)$ and $M_{low2}(w) = M(v)$ (or $M_{low1}(w) = M(v)$ and $M_{low2}(w) = M(u)$), and $high(u) < w$, $high(v) < w$, and $b_count(w) = b_count(u) + b_count(v)$.*

Then, as an implication of the following lemma, we see that the pair $\{u, v\}$ with the property that u and v are descendants of w , but are not related as ancestor and descendant, and $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut, is uniquely determined by w (and thus the number of those 3-cuts is $O(n)$).

■ **Algorithm 1** Find all 3-cuts $\{(u, p(u)), (v, p(v)), e)\}$, where u is a descendant of v and $B(v) = B(u) \sqcup \{e\}$, for a back-edge e .

```

1 initialize an array currentVertex with  $n$  entries
  //  $m = \tilde{M}(v)$ 
2 foreach vertex  $x$  do currentVertex $[x] \leftarrow x$ 
3 for  $v \leftarrow n$  to  $v = 1$  do
4    $m \leftarrow \tilde{M}(v)$ 
5   if  $m = \emptyset$  then continue
    // find the lowest  $u \in M^{-1}(m)$  which is greater than  $v$ 
6    $u \leftarrow \text{currentVertex}[m]$ ,  $\text{prev} \leftarrow u$ 
7   while  $\text{nextM}(u) \neq \emptyset$  and  $\text{nextM}(u) > v$  do  $\text{prev} \leftarrow u$ ,  $u \leftarrow \text{nextM}(u)$ 
8   currentVertex $[m] \leftarrow \text{prev}$ 
    // check the condition in Lemma 9
9   if  $\text{high}(u) < v$  and  $b\_count(v) = b\_count(u) + 1$  then
10    | mark the triplet  $\{(u, p(u)), (v, p(v)), (M(v), l(M(v)))\}$ 
11  end
12 end
  //  $m = M_{low1}(v)$ 
13 foreach vertex  $x$  do currentVertex $[x] \leftarrow x$ 
14 for  $v \leftarrow n$  to  $v = 1$  do
15    $m \leftarrow M_{low1}(v)$ 
16   if  $m = \emptyset$  or  $l(M(v)) < v$  then continue
    // find the lowest  $u \in M^{-1}(m)$  which is greater than  $v$ 
17    $u \leftarrow \text{currentVertex}[m]$ ,  $\text{prev} \leftarrow u$ 
18   while  $\text{nextM}(u) \neq \emptyset$  and  $\text{nextM}(u) > v$  do  $\text{prev} \leftarrow u$ ,  $u \leftarrow \text{nextM}(u)$ 
19   currentVertex $[m] \leftarrow \text{prev}$ 
    // check the condition in Lemma 9
20   if  $\text{high}(u) < v$  and  $b\_count(v) = b\_count(u) + 1$  then
21    | mark the triplet  $\{(u, p(u)), (v, p(v)), (M_{low2}(v), l(M_{low2}(v)))\}$ 
22  end
23 end
  //  $m = M_{low2}(v)$ 
24 foreach vertex  $x$  do currentVertex $[x] \leftarrow x$ 
25 for  $v \leftarrow n$  to  $v = 1$  do
26    $m \leftarrow M_{low2}(v)$ 
27   if  $m = \emptyset$  or  $l(M(v)) < v$  then continue
    // find the lowest  $u \in M^{-1}(m)$  which is greater than  $v$ 
28    $u \leftarrow \text{currentVertex}[m]$ ,  $\text{prev} \leftarrow u$ 
29   while  $\text{nextM}(u) \neq \emptyset$  and  $\text{nextM}(u) > v$  do  $\text{prev} \leftarrow u$ ,  $u \leftarrow \text{nextM}(u)$ 
30   currentVertex $[m] \leftarrow \text{prev}$ 
    // check the condition in Lemma 9
31   if  $\text{high}(u) < v$  and  $b\_count(v) = b\_count(u) + 1$  then
32    | mark the triplet  $\{(u, p(u)), (v, p(v)), (M_{low1}(v), l(M_{low1}(v)))\}$ 
33  end
34 end

```

► **Lemma 13.** *Let $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ be a 3-cut such that u and v are not related as ancestor and descendant and let w is an ancestor of both u and v . By Proposition 11 and Lemma 12, we may assume w.l.o.g. that $M_{low1}(w) = M(u)$ and $M_{low2}(w) = M(v)$, and let $m_1 = M_{low1}(w)$ and $m_2 = M_{low2}(w)$. Then u is the lowest vertex in $M^{-1}(m_1)$ which is greater than w , and v is the lowest vertex in $M^{-1}(m_2)$ which is greater than w .*

This suggests a method to compute those u, v (if they exist) for a particular w . We simply have to find the lowest u in $M^{-1}(M_{low1}(w))$ which is greater than w , and the lowest v in $M^{-1}(M_{low2}(w))$ which is greater than w , and, if they exist, check whether $high(u) < w$, $high(v) < w$, and $b_count(w) = b_count(u) + b_count(v)$. To perform this search efficiently, we have the lists $M^{-1}(x)$, for every vertex x , sorted in decreasing order, we process the vertices w in a bottom-up fashion, and we keep stored in a variable *currentVertex*[x] the most recent element of $M^{-1}(x)$ that we considered. Algorithm 2 is an implementation of this procedure, for computing all 3-cuts of this type.

■ **Algorithm 2** Find all 3-cuts $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where w is an ancestor of u and v , and u, v are not related as ancestor and descendant.

```

1 initialize an array currentVertex with  $n$  entries
2 foreach vertex  $x$  do currentVertex[ $x$ ]  $\leftarrow x$ 
3 for  $w \leftarrow n$  to  $w = 1$  do
4    $m_1 \leftarrow M_{low1}(w)$ ,  $m_2 \leftarrow M_{low2}(w)$ 
5   if  $m_1 = \emptyset$  or  $m_2 = \emptyset$  then continue
6   // find the lowest  $u$  in  $M^{-1}(m_1)$  which is greater than  $w$ 
7    $u \leftarrow \text{currentVertex}[m_1]$ 
8   while  $\text{nextM}(u) \neq \emptyset$  and  $\text{nextM}(u) > w$  do  $u \leftarrow \text{nextM}(u)$ 
9   currentVertex[ $m_1$ ]  $\leftarrow u$ 
10  // find the lowest  $v$  in  $M^{-1}(m_2)$  which is greater than  $w$ 
11   $v \leftarrow \text{currentVertex}[m_2]$ 
12  while  $\text{nextM}(v) \neq \emptyset$  and  $\text{nextM}(v) > w$  do  $v \leftarrow \text{nextM}(v)$ 
13  currentVertex[ $m_2$ ]  $\leftarrow v$ 
14  // check the condition in Lemma 12
15  if  $b\_count(w) = b\_count(u) + b\_count(v)$  and  $high(u) < w$  and  $high(v) < w$ 
16    then
17      | mark the triplet  $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ 
18    end
19 end

```

Now we treat the case that u and v are related as ancestor and descendant, and assume w.l.o.g. that u is a descendant of v . We have the following characterization of those 3-cuts.

► **Proposition 14.** *Let u, v, w be three vertices such that u is a descendant of v and v is a descendant of w . Then $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut if and only if $B(v) = B(u) \sqcup B(w)$.*

This implies that $M(v)$ is an ancestor of $M(w)$, and we distinguish two cases, depending on whether $M(v)$ is a proper ancestor of $M(w)$. In the first case we have the following.

► **Lemma 15.** *Let u be a descendant of v and v a descendant of w , and $M(v) \neq M(w)$. Then, $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut if and only if: $M(w) = M_{low1}(v)$ and w is the greatest vertex with $M(w) = M_{low1}(v)$ which is lower than v , $M(u) = M_{low2}(v)$ and u*

47:10 Computing the 4-Edge-Connected Components of a Graph in Linear Time

is the lowest vertex with $M(u) = M_{low2}(v)$, $high(u) < v$ and $b_count(v) = b_count(u) + b_count(w)$.

This shows that the number of such 3-cuts is $O(n)$, and it immediately suggests an algorithm to compute them efficiently (i.e. we work as in Algorithm 2).

Now, if $M(v) = M(w)$, we distinguish two cases, depending on whether $w = nextM(v)$ or $w < nextM(v)$. In any case, there is a unique u which is a descendant of v such that $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut, since by Proposition 14 we have $B(u) = B(v) \setminus B(w)$, and we have assumed that the graph is 3-edge-connected (and so the result follows from Fact 3). The next lemma shows that u satisfies $high(u) = high(v)$ and $nextM(u) = \emptyset$.

► **Lemma 16.** *Let u, v, w be three vertices such that u is a descendant of v , v is a descendant of w , and $M(v) = M(w)$. Then, $B(v) = B(u) \sqcup B(w)$ only if $high(u) = high(v)$ and $nextM(u) = \emptyset$.*

Thus, for every vertex h , we seek in the decreasingly sorted list $high^{-1}(h)$ pairs of vertices $\{u, v\}$ that have the potential to provide a 3-cut of the form $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where u is a descendant of v , v is a descendant of w , and $M(v) = M(w)$. In the case $w = nextM(v)$ we have the following:

► **Proposition 17.** *Let $h = high(v)$ and $w = nextM(v)$, and suppose that the list $high^{-1}(h)$ is sorted in decreasing order. Then, u is a descendant of v such that $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut if and only if u is a predecessor of v in $high^{-1}(h)$, $nextM(u) = \emptyset$, $low(u) \geq w$, $b_count(u) = b_count(v) - b_count(w)$, and all elements of $high^{-1}(h)$ between u and v are ancestors of u .*

Thus we traverse the decreasingly sorted list $high^{-1}(h)$ from its first element, and we keep consecutive entries that are related as ancestor and descendant in a stack. When we meet a $v \in high^{-1}(h)$ that has $nextM(v) \neq \emptyset$, we simply check whether there is an entry u in the stack that satisfies $nextM(u) = \emptyset$, $low(u) \geq nextM(v)$ and $b_count(u) = b_count(v) - b_count(nextM(v))$, whence we immediately infer that $\{(u, p(u)), (v, p(v)), (nextM(v), p(nextM(v)))\}$ is a 3-cut. This procedure is shown in Algorithm 3.

The case $w < nextM(v)$ is more complicated, since for a particular $v \in high^{-1}(h)$ there may be many pairs $\{u, w\}$ such that u is a descendant of v and w is a proper ancestor of $nextM(v)$ with $M(w) = M(v)$, and $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut. Thus, we keep in a stack $stackU[v]$, for every $v \in high^{-1}(h)$, a set of $u \in high^{-1}(h)$ with the potential to provide such a 3-cut. In particular, let $\tilde{U}(v)$, for a vertex v , denote the set of all descendants u of v with the property that there exists a w with $M(w) = M(v)$ and $w < nextM(v)$, such that $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut. Then the stacks $stackU[v]$ are filled with Algorithm 4, and satisfy the following:

► **Lemma 18.** *For every vertex v we have $\tilde{U}(v) \subseteq stackU(v)$, and for every $v' \neq v$ we have $stackU(v) \cap stackU(v') = \emptyset$. Furthermore, if u' is a successor of u in $stackU(v)$, then u' is an ancestor of u .*

This implies that the number of 3-cuts of the form $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where u is a descendant of v and w is a proper ancestor of $nextM(v)$ with $M(w) = M(v)$, is $O(n)$. The next lemma provides a criterion to determine whether a $u \in stackU(v)$ is in $\tilde{U}(v)$, and a way to compute the vertex w with $M(w) = M(v)$ and $w < nextM(v)$, such that $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut.

■ **Algorithm 3** Find all 3-cuts $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where u is a descendant of v and $w = \text{nextM}(v)$.

```

1 initialize an array  $A$  with  $m$  entries (where  $m$  is the number of edges of the graph)
2 initialize a stack  $S$ 
3 sort the elements of every list  $\text{high}^{-1}(h)$ , for every vertex  $h$ , in decreasing order
4 foreach vertex  $h$  do
5    $u \leftarrow$  first element of  $\text{high}^{-1}(h)$ 
6   while  $u \neq \emptyset$  do
7      $z \leftarrow$  next element of  $\text{high}^{-1}(h)$ 
8     if  $z = \emptyset$  then break
9     if  $z$  is not an ancestor of  $u$  then
10      while  $S$  is not empty do
11         $u' \leftarrow S.\text{pop}()$ 
12         $A[b\_count(u')] \leftarrow \emptyset$ 
13      end
14    end
15    if  $\text{nextM}(z) = \emptyset$  then
16       $S.\text{push}(z)$ 
17       $A[b\_count(z)] \leftarrow z$ 
18    end
19    else if  $\text{nextM}(z) \neq \emptyset$  then
20       $v \leftarrow z, w \leftarrow \text{nextM}(v)$ 
21      if  $A[b\_count(v) - b\_count(w)] \neq \emptyset$  then
22         $u \leftarrow A[b\_count(v) - b\_count(w)]$ 
23        if  $\text{low}(u) \geq w$  then
24          mark the triplet  $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ 
25        end
26      end
27    end
28     $u \leftarrow z$ 
29  end
30 end

```

► **Lemma 19.** Let u be a vertex in $\text{stackU}[v]$ and w a proper ancestor of v such that $M(w) = M(v)$. Then, if $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut, we have that $b_count(v) = b_count(u) + b_count(w)$ and w is the greatest element of $M^{-1}(M(v))$ such that $w \leq \text{low}(u)$. Conversely, if $b_count(v) = b_count(u) + b_count(w)$ and $w \leq \text{low}(u)$, then $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ is a 3-cut.

Thus, for every $u \in \text{stackU}[v]$, we have to find the greatest $w \in M^{-1}(M(v))$ such that $w \leq \text{low}(u)$ and $b_count(v) = b_count(u) + b_count(w)$. To do this efficiently, we take advantage of the fact that the stack $\text{stackU}[v]$ has been filled in such a way, that the successor of every $u \in \text{stackU}[v]$ is an ancestor of u , and of the fact that $\text{low}(u') \leq \text{low}(u)$, for every ancestor u' of u . Then we have the lists $M^{-1}(x)$, for every vertex x , sorted in decreasing order, and we process the vertices v from lowest to highest. For every $u \in \text{stackU}[v]$, we traverse the list $M^{-1}(M(v))$ in order to find the greatest $w \in M^{-1}(M(v))$ that has $w \leq \text{low}(u)$.

■ **Algorithm 4** Fill all stacks $stackU[v]$, for all vertices v .

```

1 initialize a stack  $S$ 
2 sort the elements of every list  $high^{-1}(h)$ , for every vertex  $h$ , in decreasing order
3 foreach vertex  $v$  do initialize a stack  $stackU[v]$ 
4 foreach vertex  $h$  do
5    $u \leftarrow$  first element of  $high^{-1}(h)$ 
6   while  $u \neq \emptyset$  do
7      $z \leftarrow$  next element of  $high^{-1}(h)$ 
8     if  $z = \emptyset$  then break
9     if  $z$  is not an ancestor of  $u$  then
10      | pop out all elements from  $S$ 
11    end
12    if  $nextM(z) = \emptyset$  then
13      |  $S.push(z)$ 
14    end
15    else if  $nextM(z) \neq \emptyset$  then
16      | while  $low(S.top()) < lastM(z)$  do  $S.pop()$ 
17      | while  $low(S.top()) < nextM(z)$  do
18      |    $u \leftarrow S.pop()$ 
19      |    $stackU[v].push(u)$ 
20      | end
21    end
22     $u \leftarrow z$ 
23  end
24 end

```

Using a path-compression method, we can bypass segments of $M^{-1}(M(v))$ that we have already visited. This procedure is shown in Algorithm 5. A detailed proof of correctness and linear complexity is given in the full version of this paper [6].

■ **Algorithm 5** Find all 3-cuts $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where u is a descendant of v , v is a descendant of w with $M(v) = M(w)$, and $w \neq nextM(v)$.

```

1 initialize an array  $lowestW$  with  $n$  entries
2 foreach vertex  $v$  do  $lowestW[v] \leftarrow nextM(v)$ 
3 for  $v \leftarrow 1$  to  $v \leftarrow n$  do
4   while  $stackU[v].top() \neq \emptyset$  do
5      $u \leftarrow stackU[v].pop()$ 
6      $w \leftarrow lowestW[v]$ 
7     while  $w > low(u)$  do  $w \leftarrow lowestW[w]$ 
8      $lowestW[v] \leftarrow w$ 
9     if  $b\_count(v) = b\_count(u) + b\_count(w)$  then
10      | mark the triplet  $\{(u, p(u)), (v, p(v)), (w, p(w))\}$ 
11    end
12  end
13 end

```

4 Computing the 4-edge-connected components in linear time

Now we consider how to compute the 4-edge-connected components of an undirected graph G in linear time. First, we reduce this problem to the computation of the 4-edge-connected components of a collection of auxiliary 3-edge-connected graphs.

4.1 Reduction to the 3-edge-connected case

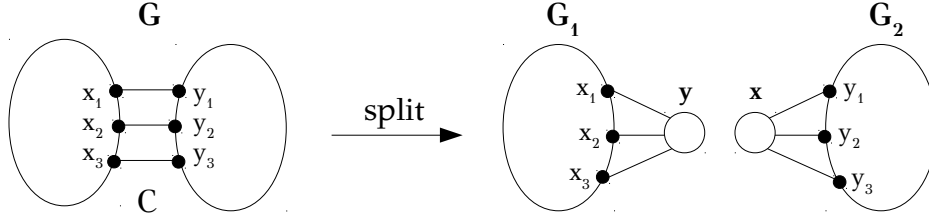
Given a (general) undirected graph G , we execute the following steps:

1. Compute the connected components of G .
2. For each connected component, we compute the 2-edge-connected components which are subgraphs of G .
3. For each 2-edge-connected component, we compute its 3-edge-connected components C_1, \dots, C_ℓ .
4. For each 3-edge-connected component C_i , we compute a 3-edge-connected auxiliary graph H_i , such that for any two vertices x and y , we have $x \stackrel{G}{\equiv}_4 y$ if and only if x and y are both in the same auxiliary graph H_i and $x \stackrel{H_i}{\equiv}_4 y$.
5. Finally, we compute the 4-edge-connected components of each H_i .

Steps 1–3 take overall linear time [21, 25]. We describe step 5 in the next section, so it remains to give the details of step 4. Let H be a 2-edge-connected component (subgraph) of G . We can construct a compact representation of the 2-cuts of H , which allows us to compute its 3-edge-connected components C_1, \dots, C_ℓ in linear time [7, 25]. Now, since the collection $\{C_1, \dots, C_\ell\}$ constitutes a partition of the vertex set of H , we can form the quotient graph Q of H by shrinking each C_i into a single node. Graph Q has the structure of a tree of cycles [2]; in other words, Q is connected and every edge of Q belongs to a unique cycle. Let (C_i, C_j) and (C_i, C_k) be two edges of Q which belong to the same cycle. Then (C_i, C_j) and (C_i, C_k) correspond to two edges (x, y) and (x', y') of G , with $x, x' \in C_i$. If $x \neq x'$, we add a virtual edge (x, x') to $G[C_i]$. (The idea is to attach (x, x') to $G[C_i]$ as a substitute for the cycle of Q which contains (C_i, C_j) and (C_i, C_k) .) Now let \bar{C}_i be the graph $G[C_i]$ plus all those virtual edges. Then \bar{C}_i is 3-edge-connected and its 4-edge-connected components are precisely those of G that are contained in C_i [2]. Thus we can compute the 4-edge-connected components of G by computing the 4-edge-connected components of the graphs $\bar{C}_1, \dots, \bar{C}_\ell$ (which can easily be constructed in total linear time). Since every \bar{C}_i is 3-edge-connected, we can apply Algorithm 6 of the following section to compute its 4-edge-connected components in linear time. Finally, we define the multiplicity $m(e)$ of an edge $e \in \bar{C}_i$ as follows: if e is virtual, $m(e)$ is the number of edges of the cycle of Q which corresponds to e ; otherwise, $m(e)$ is 1. Then, the number of minimal 3-cuts of H is given by the sum of all $m(e_1) \cdot m(e_2) \cdot m(e_3)$, for every 3-cut $\{e_1, e_2, e_3\}$ of \bar{C}_i , for every $i \in \{1, \dots, \ell\}$ [2]. Since the 3-cuts of every \bar{C}_i can be computed in linear time, the minimal 3-cuts of H can also be computed within the same time bound.

4.2 Computing the 4-edge-connected components of a 3-edge-connected graph

Now we describe how to compute the 4-edge-connected components of a 3-edge-connected graph G in linear time. Let r be a distinguished vertex of G , and let C be a minimum cut of G . By removing C from G , G becomes disconnected into two connected components. We let V_C denote the connected component of $G \setminus C$ that does not contain r , and we refer to the number of vertices of V_C as the r -size of the cut C . (Of course, these notions are relative to r .)



■ **Figure 3** $C = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$ is a 3-cut of G , with $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ lying in different connected components of $G \setminus C$. The split operation of G at C consists of the removal of the edges of C from G , and the introduction of two new nodes x, y , and six virtual edges $(x_1, y), (x_2, y), (x_3, y), (x, y_1), (x, y_2), (x, y_3)$. Now, the split graph is made of two connected components, G_1 and G_2 . Every 3-cut $C' \neq C$ of G (or more precisely: a 3-cut that corresponds to C') lies entirely within G_1 or G_2 . Conversely, every 3-cut of either G_1 or G_2 corresponds to a 3-cut of G . Thus, every 4-edge-connected component of G lies entirely within G_1 or G_2 .

Let $G = (V, E)$ be a 3-edge-connected graph, and let \mathcal{C} be the collection of the 3-cuts of G . If the collection \mathcal{C} is empty, then G is 4-edge-connected, and V is the only 4-edge-connected component of G . Otherwise, let $C \in \mathcal{C}$ be a 3-cut of G . By removing C from G , G is separated into two connected components, and every 4-edge-connected component of G lies entirely within a connected component of $G \setminus C$. This observation suggests a recursive algorithm for computing the 4-edge-connected components of G , by successively splitting G into smaller graphs according to its 3-cuts. Thus, we start with a 3-cut C of G , and we perform the splitting operation shown in Figure 3. Then we take another 3-cut C' of G and we perform the same splitting operation on the part which contains (the corresponding 3-cut of) C' . We repeat this process until we have considered every 3-cut of G . When no more splits are possible, the connected components of the final split graph correspond (by ignoring the newly introduced vertices) to the 4-edge-connected components of G .

To implement this procedure in linear time, we must take care of two things. First, whenever we consider a 3-cut C of G , we have to be able to know which ends of the edges of C belong to the same connected component of $G \setminus C$. And second, since an edge e of a 3-cut of the original graph may correspond to two virtual edges of the split graph, we have to be able to know which is the virtual edge that corresponds to e . We tackle both these problems by locating the 3-cuts of G on a DFS-tree T of G rooted at r , and by processing them in increasing order with respect to their r -size. By locating a 3-cut $C \in \mathcal{C}$ on T we can answer in $O(1)$ time which ends of the edges of C belong to the same connected component of $G \setminus C$. And then, by processing the 3-cuts of G in increasing order with respect to their size, we ensure that (the 3-cut that corresponds to) a 3-cut $C \in \mathcal{C}$ that we process lies in the split part of G that contains r .

Now, due to the analysis of the preceding sections, we can distinguish the following types of 3-cuts on a DFS-tree T (see also Figure 2):

- (I) $\{(v, p(v)), (x_1, y_1), (x_2, y_2)\}$, where (x_1, y_1) and (x_2, y_2) are back-edges.
- (IIa) $\{(u, p(u)), (v, p(v)), (x, y)\}$, where u is a descendant of v and $(x, y) \in B(v)$.
- (IIb) $\{(u, p(u)), (v, p(v)), (x, y)\}$, where u is a descendant of v and $(x, y) \in B(u)$.
- (III) $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where w is an ancestor of both u and v , but u, v are not related as ancestor and descendant.
- (IV) $\{(u, p(u)), (v, p(v)), (w, p(w))\}$, where u is a descendant of v and v is a descendant of w .

Let r be the root of T . Then, for every 3-cut $C \in \mathcal{C}$, V_C is either $T(v)$, or $T(v) \setminus T(u)$, or $T(w) \setminus (T(u) \cup T(v))$, or $T(u) \cup (T(w) \setminus T(v))$, depending on whether C is of type (I), (II), (III), or (IV), respectively. Thus we can immediately calculate the size of C and the ends of its edges that lie in V_C . In particular, the size of C is either $ND(v)$, or $ND(v) - ND(u)$, or $ND(w) - ND(u) - ND(v)$, or $ND(u) + ND(w) - ND(v)$, depending on whether it is of type (I), (II), (III), or (IV), respectively; V_C contains either $\{v, x_1, x_2\}$, or $\{p(u), v, x\}$, or $\{p(u), v, y\}$, or $\{p(u), p(v), w\}$, or $\{u, p(v), w\}$, depending on whether C is of type (I), (IIa), (IIb), (III), or (IV), respectively.

Algorithm 6 shows how we can compute the 4-edge-connected components of G in linear time, by repeatedly splitting G into smaller graphs according to its 3-cuts. When we process a 3-cut C of G , we have to find the edges of the split graph that correspond to those of C , in order to delete them and replace them with (new) virtual edges. That is why we use the symbol v' , for a vertex $v \in V$, to denote a vertex that corresponds to v in the split graph. (Initially, we set $v' \leftarrow v$.) Now, if (x, y) is an edge of C with $x \in V_C$, the edge of the split graph corresponding to (x, y) is (x', y') . Then we add two new vertices v_C and \tilde{v}_C to G , and the virtual edges (x', \tilde{v}_C) and (v_C, y') . Finally, we let x correspond to v_C , and so we set $x' \leftarrow v_C$. This is sufficient, since we process the 3-cuts of G in increasing order with respect to their size, and so the next time we meet the edge (x, y) in a 3-cut, we can be certain that it corresponds to (v_C, y') .

■ **Algorithm 6** Compute the 4-edge-connected components of a 3-edge-connected graph $G = (V, E)$.

```

1 Find the collection  $\mathcal{C}$  of the 3-cuts of  $G$ 
2 Locate and classify the 3-cuts of  $G$  on a DFS-tree of  $G$  rooted at  $r$ 
3 For every  $C \in \mathcal{C}$ , calculate  $size(C)$  (relative to  $r$ )
4 Sort  $\mathcal{C}$  in increasing order w.r.t. the  $size$  of its elements
5 foreach  $v \in V$  do Set  $v' \leftarrow v$ 
6 foreach  $C = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} \in \mathcal{C}$  do
7   Find the ends of the edges of  $C$  that lie in  $V_C$  // Let those ends be  $x_1, x_2$ 
   and  $x_3$ 
8   Remove the edges  $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3)$  from  $G$ 
9   Introduce two new vertices  $v_C$  and  $\tilde{v}_C$  to  $G$ 
10  Add the edges  $(x'_1, \tilde{v}_C), (x'_2, \tilde{v}_C), (x'_3, \tilde{v}_C), (v_C, y'_1), (v_C, y'_2), (v_C, y'_3)$  to  $G$ 
11  Set  $x'_1 \leftarrow v_C, x'_2 \leftarrow v_C, x'_3 \leftarrow v_C$ 
12 end
13 Output the connected components of  $G$ , ignoring the newly introduced vertices

```

Final Remarks

Independently from our work, Nadara et al. [17] also presented a linear-time algorithm for computing the 4-edge-connected components of a graph. Both our algorithm and the algorithm of [17] require the use of the static tree disjoint-set-union data structure of Gabow and Tarjan [4] to achieve linear running time. Also, similar to our algorithm, the main part in the algorithm of Nadara et al. is the computation of the 3-edge cuts of a 3-edge-connected graph G . Both algorithms operate on a depth-first search tree of G , with start vertex r , and distinguish 3 types of cuts $C = \{e_1, e_2, e_3\}$, depending on the number of tree edges in C . These cases are handled in a different manner in [17]. In particular, Nadara et al. [17] show

that the case where C consists of three tree edges can be reduced, in linear time, to the other two cases. We note that by applying this idea in our framework, we are able to avoid the use of *high* points. (We achieve this by modifying the algorithm that identifies 3-edge cuts consisting of two tree edges, described in Section 3.2.) This way, we obtain a linear-time algorithm that does not require the Gabow-Tarjan disjoint-set-union data structure, and thus is implementable in the pointer machine computation model [24].

References

- 1 E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization (in Russian)*, page 290–306, 1976.
- 2 Y. Dinitz. The 3-edge-components and a structural description of all 3-edge-cuts in a graph. In *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '92, page 145–157, Berlin, Heidelberg, 1992. Springer-Verlag.
- 3 Y. Dinitz and J. Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20:242–276, 1998. doi:10.1007/PL00009195.
- 4 H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–21, 1985.
- 5 Z. Galil and G. F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, 1991. doi:10.1145/122413.122416.
- 6 L. Georgiadis, G. F. Italiano, and E. Kosinas. Computing the 4-edge-connected components of a graph in linear time. *CoRR*, abs/2105.02910, 2021. arXiv:2105.02910.
- 7 L. Georgiadis and E. Kosinas. Linear-Time Algorithms for Computing Twinless Strong Articulation Points and Related Problems. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2020.38.
- 8 M. Ghaffari, K. Nowicki, and M. Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1260–1279, USA, 2020. Society for Industrial and Applied Mathematics.
- 9 M. Henzinger, S. Rao, and D. Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020.
- 10 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
- 11 A. Kanevsky and V. Ramachandran. Improved algorithms for graph four-connectivity. *Journal of Computer and System Sciences*, 42(3):288–306, 1991. doi:10.1016/0022-0000(91)90004-0.
- 12 A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science (FOCS 1991)*, pages 793–801, 1991. doi:10.1109/SFCS.1991.185451.
- 13 D. R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, January 2000. doi:10.1145/331605.331608.
- 14 D. R. Karger and D. Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, page 246–255, USA, 2009. Society for Industrial and Applied Mathematics.
- 15 K.-I. Kawarabayashi and M. Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM*, 66(1), December 2018. doi:10.1145/3274663.
- 16 K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.
- 17 W. Nadara, M. Radecki, M. Smulewicz, and M. Sokolowski. Determining 4-edge-connected components in linear time. In *Proc. 29th European Symposium on Algorithms*, 2021.

- 18 H. Nagamochi and T. Ibaraki. A linear time algorithm for computing 3-edge-connected components in a multigraph. *Japan J. Indust. Appl. Math*, 9(163), 1992. doi:10.1007/BF03167564.
- 19 H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition.
- 20 H. Nagamochi and T. Watanabe. Computing k -edge-connected components of a multigraph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 76(4):513–517, 1993.
- 21 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- 22 R. E. Tarjan. Finding dominators in directed graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974.
- 23 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
- 24 R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–27, 1979.
- 25 Y. H. Tsin. Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130–146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP). doi:10.1016/j.jda.2008.04.003.

Deep Multilevel Graph Partitioning

Lars Gottesbüren

Karlsruhe Institute of Technology, Germany

Tobias Heuer

Karlsruhe Institute of Technology, Germany

Peter Sanders

Karlsruhe Institute of Technology, Germany

Christian Schulz

Universität Heidelberg, Germany

Daniel Seemaier

Karlsruhe Institute of Technology, Germany

Abstract

Partitioning a graph into blocks of “roughly equal” weight while cutting only few edges is a fundamental problem in computer science with a wide range of applications. In particular, the problem is a building block in applications that require parallel processing. While the amount of available cores in parallel architectures has significantly increased in recent years, state-of-the-art graph partitioning algorithms do not work well if the input needs to be partitioned into a large number of blocks. Often currently available algorithms compute highly imbalanced solutions, solutions of low quality, or have excessive running time for this case. This is due to the fact that most high-quality general-purpose graph partitioners are *multilevel algorithms* which perform graph coarsening to build a hierarchy of graphs, initial partitioning to compute an initial solution, and local improvement to improve the solution throughout the hierarchy. However, for large number of blocks, the smallest graph in the hierarchy that is used for initial partitioning still has to be large.

In this work, we substantially mitigate these problems by introducing *deep multilevel graph partitioning* and a shared-memory implementation thereof. Our scheme continues the multilevel approach deep into initial partitioning – integrating it into a framework where recursive bipartitioning and direct k -way partitioning are combined such that they can operate with high performance and quality. Our integrated approach is stronger, more flexible, arguably more elegant, and reduces bottlenecks for parallelization compared to existing multilevel approaches. For example, for large number of blocks our algorithm is on average at least an order of magnitude faster than competing algorithms while computing partitions with comparable solution quality. At the same time, our algorithm consistently produces balanced solutions. Moreover, for small number of blocks, our algorithms are the fastest among competing systems with comparable quality.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph partitioning, graph algorithms, multilevel, shared-memory, parallel

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.48

Supplementary Material The source code and data has been made available at https://algo2.iti.kit.edu/seemaier/deep_mgp/ as well as <https://github.com/KaHIP/KaMinPar>.

Funding The authors acknowledge support by the State of Baden-Württemberg through bwHPC. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 882500).



© Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 48; pp. 48:1–48:17

Leibniz International Proceedings in Informatics



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

1 Introduction

Graphs are a universal abstraction for modelling relations between objects. Thus they are used throughout computer science and have applications with an ever growing volume and variety of the considered graphs. One frequently needed basic operation is *balanced graph partitioning* – cutting a graph into k pieces of “roughly equal” weight while cutting only few edges. Balanced graph partitioning is NP-hard and even NP-hard to approximate [5] and thus usually solved using heuristics. In particular, *multilevel graph partitioning (MGP)* is used in most high-quality general-purpose systems: During *coarsening*, build a hierarchy of graphs where each graph is a coarse approximation of the previous one. When the coarse graph is “small”, run a possibly expensive *initial partitioning* method on it. This is useful because a feasible partition at the coarsest level is a feasible partition of the original input with the same cut value. The partition is successively *uncoarsened* to each finer level and *locally improved*. This is often both faster and higher quality than applying comparable improvement algorithms only on the finest level since MGPs have a more global view on the coarse levels and can move entire groups of nodes in constant time.

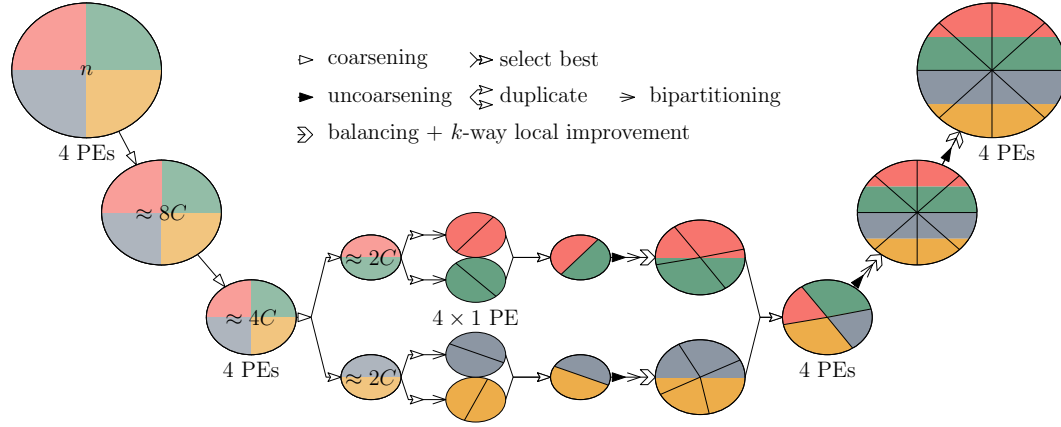
A prominent application (out of many) is distributing workload across parallel machines with little communication. With growing numbers of processors in parallel machines, we are interested in large values of k – in the order of millions. However, existing research has mostly focused on small values, typically $2 \leq k \leq 256$. Unsurprisingly, these systems perform poorly for large k . If *direct k -way partitioning* is used, the coarsest graph still has to be large when initial partitioning is called. *Recursive bipartitioning* performs $\log(k)$ cycles of (un)coarsening and is either restricted to very small imbalances or unlikely to return a feasible solution. Further, both exhibit parallelization bottlenecks on coarse levels.

We mitigate these problems by introducing *deep MGP*, an approach that continues the multilevel scheme deep into initial partitioning and integrates aspects of direct k -way and recursive bipartitioning. Deep MGP can be instantiated with concrete (parallel) building blocks for (un)coarsening, k -way local improvement, and bipartitioning of small graphs. We also include *balancing* as an explicit component. Figure 1 summarizes the approach. Deep MGP performs only one cycle of (un)coarsening. Bipartitioning is done during uncoarsening so that it is always applied to graphs with about C nodes (input parameter) until the desired number of k blocks is reached. To exploit all the available parallelism, the invariant is maintained that parallel tasks performed by x processing elements (*PEs*) work on graphs with at least xC nodes. Maintaining this invariant during coarsening allows multiple diversified attempts with little overhead invested.

Under certain simplifying assumptions, deep MGP for k -partitioning an n -node graph with bounded degree can be done in time $\mathcal{O}((n/p) \max(1, \log(kC/n)) + \log^2 n)$, i.e., with linear work and polylogarithmic span unless k is very large; see Section 4.

After introducing notation and basic concepts in Section 2 and discussing related work in Section 3, Section 4 introduces deep MGP as a generic method. In Section 5 we describe the simple and fast **KaMinPar** partitioner which uses deep MGP to achieve scalability to both large k and a considerable number of parallel cores while guaranteeing balanced solutions.

In Section 6 we report on extensive experiments which indicate that **KaMinPar** has a very favorable quality-performance tradeoff and very good scaling behavior. For traditional values of k , it is faster than previous algorithms that can achieve comparable or better quality. For large k , previous algorithms mostly find infeasible solutions and exhibit excessive running times, whereas **KaMinPar** consistently finds feasible solutions with comparable quality and is an order of magnitude faster. Section 7 summarizes the results and outlines possible future directions.



■ **Figure 1** Partitioning a graph with n nodes into 8 blocks using 4 PEs. Duplicate while coarsening to maintain load $\geq C$ on each PE. Successively bipartition during uncoarsening. Colors indicate work performed by each PE.

2 Preliminaries

Notation and Definitions. Let $G = (V, E, c, \omega)$ be an undirected graph with node weights $c : V \rightarrow \mathbb{N}_{>0}$, edge weights $\omega : E \rightarrow \mathbb{N}_{>0}$, $n := |V|$, and $m := |E|$. We extend c and ω to sets, i.e., $c(V') := \sum_{v \in V'} c(v)$ and $\omega(E') := \sum_{e \in E'} \omega(e)$. $N(v) := \{u \mid \{u, v\} \in E\}$ denotes the neighbors of v and $E(v) := \{e \mid v \in e\}$ denotes the edges incident to v . For some $V' \subseteq V$, $G[V']$ denotes the subgraph of G induced by V' . We are looking for *blocks* of nodes $\Pi := \{V_1, \dots, V_k\}$ that partition V , i.e., $V_1 \cup \dots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The *balance constraint* demands that $\forall i \in \{1..k\} : c(V_i) \leq L_{\max, k} := \max\{(1 + \varepsilon) \frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v)\}$ for some imbalance parameter ε .¹ The objective is to minimize $\text{cut}(\Pi) := \sum_{i < j} \omega(E_{ij})$ (weight of all cut edges), where $E_{ij} := \{\{u, v\} \in E \mid u \in V_i, v \in V_j\}$. We call a node $v \in V_i$ that has a neighbor $w \in V_j$, $i \neq j$ a *boundary node*. A *clustering* $\mathcal{C} := \{C_1, \dots, C_l\}$ is also a partition of V , where the number of blocks l is not given in advance (there is also no balance constraint).

Multilevel Graph Partitioning. Many high-quality graph partitioners employ the multilevel paradigm, which consists of three phases: During the *coarsening phase*, the algorithms build a hierarchy of successively smaller graphs where each graph is a coarse approximation of the previous one. Coarse graphs are built by either computing node clusters or matchings and afterwards *contracting* them. A clustering $\mathcal{C} = \{C_1, \dots, C_l\}$ is contracted by collapsing each cluster C_i into a single node c_i with weight $c(c_i) = \sum_{v \in C_i} c(v)$. There is an edge $e = (c_i, c_j)$ in the contracted graph with weight $\omega(e) = \sum_{(u, v) \in E_{ij}} \omega(u, v)$ where E_{ij} are the edges that connect cluster C_i and C_j in the original graph, if $|E_{ij}| > 0$. Once the number of nodes of a coarse graph falls below a certain threshold or the coarsening algorithm converges, *initial partitioning* computes a partition of the coarsest graph. Finally, *refinement* subsequently undoes the contractions performed during coarsening. In each uncontraction, the partition is first projected to the finer graph and then improved using *local improvement* algorithms.

¹ Traditionally, $L_k := (1 + \varepsilon) \lceil \frac{c(V)}{k} \rceil$ is used as balance constraint. However, finding a balanced partition with L_k is NP-complete, which, as we will see in Section 4, is not case for $L_{\max, k}$.

Generally, there are two ways to partition a graph into k blocks using the multilevel paradigm, namely *direct k -way* partitioning and *recursive bipartitioning*. The former coarsens the graph until $\Omega(k)$ nodes are left – usually kC nodes where C is an input parameter – and then computes a k -way partition of the coarsest graph. The latter first computes a bipartition $\Pi = \{V_1, V_2\}$ and then recurses on the induced subgraphs $G[V_1]$ and $G[V_2]$ by partitioning V_1 into $\lceil \frac{k}{2} \rceil$ and V_2 into $\lfloor \frac{k}{2} \rfloor$ blocks. Note that many multilevel graph partitioners based on the direct k -way partitioning scheme use recursive bipartitioning to compute an initial partition of the coarsest graph [25, 37, 28, 3, 19].

Size-Constrained Label Propagation. Based on the *label propagation clustering* algorithm by Raghavan et al. [36], Meyerhenke et al. [34] introduced the *size-constrained label propagation* algorithm as a coarsening and refinement algorithm. The algorithm is parameterized by a maximum cluster size U . In the coarsening resp. refinement phase, each node is initially assigned to its own cluster resp. to its corresponding block of the partition. The algorithm then works in rounds. In each round, the nodes are visited in some order and a node u is moved to the cluster resp. block K that contains the most neighbors of u without violating the size constraint, i.e., $c(K) + c(u) \leq U$. The algorithm terminates once no more nodes were moved during a round or a maximum number of rounds has been exceeded.

Maintaining the Balance Constraint. Finding a balanced partition of a weighted graph with $L_k := (1 + \varepsilon) \lceil \frac{c(V)}{k} \rceil$ as balance constraint is NP-complete as it can be reduced to the problem of scheduling jobs on identical parallel machines [17]. Therefore, many partitioners incorporate techniques that prevent the formation of heavy nodes during the coarsening process by penalizing the contraction of nodes with large weights [9] or enforcing a strict upper bound for node weights [21]. This makes it easier for initial partitioning to find a feasible initial solution. However, as we will see in Section 4, if we replace L_k with $L_{\max, k}$ the problem of finding a balanced partition becomes solvable in polynomial time. In the recursive bipartitioning setting, using the input imbalance parameter ε for each bipartition can produce blocks in the final k -way partition that would violate the balance constraint. Therefore, KaHyPar [20, 38] ensures that a k -way partition obtained via recursive bipartitioning is balanced by adapting the imbalance ratio for each bipartition individually. Let $G[V']$ be the subgraph of the current bipartition that should be partitioned recursively into $k' \leq k$ blocks. Then, $\varepsilon' := \left((1 + \varepsilon) \frac{c(V)}{k} \cdot \frac{k'}{c(V')} \right)^{\frac{1}{\lceil \log_2(k') \rceil}} - 1$ is the imbalance ratio used for the bipartition of $G[V']$. If each bipartition is ε' -balanced, then the final k -way partition is ε -balanced.

3 Related Work

There has been a huge amount of research on graph partitioning so that we refer the reader to overview papers [39, 7, 43, 8, 40] for most of the general material. Here, we give a brief overview of techniques used in parallel multilevel graph partitioners and issues closely related to our main contributions.

Most modern high-quality graph partitioners are based on the multilevel paradigm. Well-known software packages based on this approach include KaHiP [37] and Metis [24] (sequential graph partitioners), Mt-KaHiP [4] and Mt-Metis [28, 30] (shared-memory graph partitioners), Jostle [43], PT-Scotch [11], ParHiP [35] and ParMetis [23] (distributed graph partitioners).

The matching [11, 13, 28, 23, 43] and clustering algorithms [4, 10, 19] used by different sequential partitioners in the coarsening phase are well-suited for parallelization, sometimes with only minor quality losses [10, 33]. The coarsening phase proceeds until a fixed number of nodes remains, usually kC , where C is a tuning parameter.

In the initial partitioning phase, parallel partitioners either call sequential multilevel algorithms with different random seeds [4, 11, 13, 22] or use parallel recursive bipartitioning [19, 28, 23, 30]. In the former case, the graph is copied to each PE and the best partition obtained from all independent runs is projected back to the coarsest graph. In the latter case, parallelism is achieved by either splitting the thread pool into two evenly-sized groups and assigning each to one of the two recursive calls [23, 28] or dynamically assign the threads to the recursive calls with a task scheduler [19]. To obtain an initial bipartition of the coarsest graph, often portfolio approaches composed of different bipartitioning algorithms are used [37, 38, 19, 9].

Most parallel partitioners use the label propagation heuristic to improve the solution quality during the refinement phase [4, 28, 19, 35, 43, 13]. More advanced techniques are based on parallel variants of the FM local search [15] that are widely used in sequential partitioners. PT-Scotch [11], KaPPa [22] and Jostle [43] use sequential 2-way FM refinement on two adjacent blocks of the partition. Mt-KaHiP [4] and Mt-KaHyPar [19] implement a parallel k -way FM algorithm based on the *localized multi-try* FM of the sequential graph partitioner KaHiP [37]. Mt-Metis [30] uses greedy refinement (FM with only positive gain moves), and hill-scanning, a simplification of localized FM where small groups of vertices, whose individual gains are negative, are moved if their combined gain is positive.

In the parallel setting, nodes can change their block concurrently which requires synchronization to ensure that the balance constraint is not violated [28]. Existing systems either explicitly communicate their local changes and reject moves that would violate the balance constraint [28, 23, 13] or use atomic *compare-and-swap* instructions to maintain block weights [19, 4].

Limitations of Existing Systems for Large k . The coarsening phase of MGP usually stops when kC nodes are left. For large k , this breaks the assumption that the coarsest graph is small. Thus, really expensive initial partitioners are infeasible at this level. Many MGPs therefore use multilevel recursive bipartitioning for the coarsest graph [25, 37, 28, 3, 19]. This results in a sequential running time of $\mathcal{O}(T \log k)$ where T is the running time of the bipartitioning algorithm. When this is used within a parallel algorithm, initial partitioning can become a major bottleneck [4].

Furthermore, a feasible solution for the k -way graph partitioning problem must satisfy the balance constraint that usually depends on the average block weight $\frac{c(V)}{k}$. Thus, increasing the number of blocks leads to a tighter balance constraint and drastically reduces the space of feasible solutions. Therefore, a partitioner handling larger values of k has to employ techniques to ensure that the balance constraint is not violated.

4 Generic Deep MGP

In this section, we present our first major contribution – a new partitioning scheme that we call *Deep Multilevel Graph Partitioning*. Deep MGP continues coarsening deep into the initial partitioning phase. Roughly speaking, it starts by coarsening the input graph until $2C$ nodes are left – for some input parameter C . Then, the coarsest graph is bipartitioned into two blocks. During uncoarsening, it maintains the key invariant that the partition of a coarse graph with n' nodes has $k' = \min\{k, \text{ceil}_2(\frac{n'}{C})\}$ blocks, where $\text{ceil}_2(x)$ is x rounded up to the next power of two. This value is chosen such that each invocation of flat bipartitioning works on a graph with roughly $2C$ nodes. Thus, the graph is divided into $\min\{k, \text{ceil}_2(\frac{n}{C})\}$ blocks after unrolling the graph hierarchy. If $k > \text{ceil}_2(\frac{n}{C})$, blocks are further subdivided until k blocks are obtained.

Algorithm 1 partition.

Input: $G = (V, E)$, $k, \varepsilon > 0$, const. C
Output: k' -way partition Π of G

```

1 if  $|V(G)| > 2C$  and coarsening has not
   converged then // recursion
2    $G_c := \text{coarsen}(G)$ 
3    $\Pi_c := \text{partition}(G_c, k, \varepsilon, C)$ 
4    $\Pi := \text{project}(G, \Pi_c)$ 
5 else // base case
6    $\Pi := \{V\}$ 
7  $k' := \begin{cases} k, & |V| = n \\ \text{ceil}_2(|V|/C), & \text{else} \end{cases}$ 
8  $k' := \max\{\min\{k, k'\}, 2\}$ 
9 while  $|\Pi| < k'$  do // obtain  $k'$  blocks
10   $\Pi := \text{bipartitionBlocks}(G, \Pi, k)$ 
11  $\Pi := \text{refine}(G, \text{balance}(G, \Pi))$ 
12 return  $\Pi$ 

```

Algorithm 2 bipartitionBlocks.

Input: G , k' -way partition Π , k , const. R
Output: $2k'$ -way partition Π'

```

1  $\Pi' := \emptyset$ 
2 foreach  $V_i \in \Pi$  do
3    $\varepsilon' := \left( (1 + \varepsilon)^{\frac{c(V)}{|\Pi|c(V_i)}} \right)^{1/\log_2(k/|\Pi|)} - 1$ 
4    $G_i := G[V_i]$ 
5   // compute  $R$  bipartitions of  $V_i$ 
6    $\Pi_1, \dots, \Pi_R := \text{bipartition}(G_i, \varepsilon', R)$ 
7   // select lowest (feasible) edge cut
8    $\Pi' := \Pi' \cup \text{lowest}(G_i, \Pi_1, \dots, \Pi_R)$ 

```

This approach combines the merits of direct k -way partitioning and recursive bipartitioning: similar to direct k -way partitioning, deep MGP coarsens and uncoarsens the graph only once, and enables the use of k -way local improvement algorithms throughout the graph hierarchy. Moreover, it enforces that (possibly expensive) bipartitioning algorithms are only applied to small graphs. Thereby, it eliminates the initial partitioning bottleneck for large values of k or parallel graph partitioning.

In the remainder of this section, we give a detailed description of deep MGP. We simplify this description by restricting k to powers of two, but lift this restriction in a subsequent paragraph. Finally, we describe how to parallelize it and analyze its running time.

Deep Multilevel Graph Partitioning. Deep MGP starts by coarsening the input graph $G_1 = (V_1, E_1)$, building a hierarchy of successively coarse representations G_1, \dots, G_ℓ of G_1 . This is achieved by clustering each graph G_i and contracting all clusters to build G_{i+1} . Coarsening stops once the coarsest graph G_ℓ has at most $2C$ nodes, or the process converged. In Algorithm 1, this process is implemented in Lines 2–3.

From here, we start a sequence of the following operations: use recursive initial bipartition to subdivide the current graph into more blocks, possibly rebalance the partition and improve it using a k -way local improvement algorithm, project the partition onto the previous graph $G_{\ell-1}$ and repeat the process on that graph. During these operations, we maintain the following key invariants:

(P) A coarse graph G_i is partitioned into $k_i := \text{ceil}_2(|V_i|/C)$ blocks (bounded by 2 and k).

(B) A k_i -way partition of G_i fulfills the balance constraint.

An idealized coarsening algorithm produces a graph hierarchy where the number of nodes is halved between two levels and the coarsest graph has $2C$ nodes. In this case, it is sufficient to bipartition the coarsest graph G_ℓ once to fulfill invariant (P). To restore the invariant after uncoarsening (doubling the number of nodes in the current graph), each block of the current partition has to be bipartitioned once.

In the more general case, where coarsening can shrink the number of nodes of a graph by a larger factor than 2, we use recursive initial bipartitioning to maintain (P). More precisely, whenever the partition Π_i of graph G_i violates invariant (P), we recursively bipartition each

block of Π_i until we have k_i blocks in total. In Algorithm 1, this process is implemented in Lines 7–10. Initial bipartitioning is implemented in Algorithm 2.

Uncoarsening and initial bipartitioning can cause violations of invariant (B) (see below). If this is the case, we run a balancing algorithm afterwards. The resulting partition – which satisfies invariants (P) and (B) – is then improved using a k -way local improvement algorithm (Algorithm 1, Line 11).

If $k > \text{ceil}_2(|V_1|/C)$, the partition computed by the process described above has less than k blocks. In this case, we perform an additional round of recursive initial bipartitioning to obtain k blocks, and run balancing and k -way local improvement once more on the final partition.

Parallelization. We parallelize the partitioning method described above using parallel coarsening, local improvement and balancing algorithms. On very coarse levels, we maintain the invariant that parallel tasks performed by p PEs work on graphs with at least pC nodes. This is achieved by running initial partitioning on more and more copies of the coarsened graph, as illustrated by Figure 1. We diversify this search by using randomized coarsening, initial bipartitioning and local improvement algorithms. More precisely, we follow the algorithm described above until the coarsest graph G_C has pC nodes left. To uphold the invariant that tasks performed by p PEs work on graphs with at least pC nodes, we obtain two copies G_C^r and G_C^ℓ of G_C , and split PEs into two groups (conceptually) with $p' = \frac{p}{2}$ PEs each. If $p' > 1$, we continue by coarsening G_C^r with PEs of the first group and G_C^ℓ with PEs of the second group, until each graph has $p'C$ nodes left. We proceed in this fashion recursively, until we have obtained p graphs with $2C$ nodes each after $\log_2(p)$ recursion levels.

Each of these graphs is then bipartitioned using a single PE. Let G_C^r and G_C^ℓ with respective bipartitions Π_C^r and Π_C^ℓ be two such graphs that are copies of G_C on the previous recursion level. We use the better bipartition of Π_C^r and Π_C^ℓ (i.e., if only one of these partitions is feasible, we use that one, otherwise the one with the lower edge cut) as partition Π_C of G_C . We proceed on G_C as before, i.e., bipartition each block of Π_C if applicable, and apply the balancing and local improvement algorithm. This process is repeated for all $\log_2(p)$ recursion levels.

Handling General k . The simplified description of deep MGP only considers the case where k is a power of two. For the general case, we associate each block B with a final block count f_B – the number of blocks B is subdivided into in the final partition. Initially, $f_V = k$. To bipartition B into two blocks B_0 and B_1 , we set $f_{B_0} = \lfloor \frac{f_B}{2} \rfloor$ and $f_{B_1} = \lceil \frac{f_B}{2} \rceil$, and divide the weight of B in a f_{B_0} to f_{B_1} ratio between B_0 and B_1 . Thus, once we have computed a $k' := \text{floor}_2(k)$ -way partition, there are $k - k'$ heavy blocks with $f_B = 2$ and $2k' - k$ light blocks with $f_B = 1$. During the next and final initial partitioning step, we obtain a k -way partition by only bipartitioning heavy blocks.

Maintaining the Balance Constraint. Since MGP implementations usually employ coarsening algorithms that do not guarantee strictly uniform node weights, maintaining the balance constrain used in other partitioning systems, $L_k := (1 + \varepsilon) \lceil \frac{c(V)}{k} \rceil$, becomes an NP-complete problem [17] on coarse levels. To mitigate this problem, we use $L_{\max,k} := \max\{(1 + \varepsilon) \frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v)\}$ as balance constraint instead. This ensures that a feasible partition always exists, and that it can be found with simple greedy algorithms. Both claims are based on the fact that the average block weight of a partition is $\frac{c(V)}{k}$ and thus, there always exists a block V_i with $c(V_i) \leq \frac{c(V)}{k}$. In the multilevel setting, projecting a partition

to a finer graph can violate the balance constraint due to the change in $\max_v c(v)$. However, the overload per block is bounded by $\max_v c(v)$, which implies that a balancing algorithm only needs to move a small number of nodes out of a block to restore the balance constraint.

Running Time. Next, we analyze the running time of parallel (deep) MGP using highly idealized assumption. We do not claim that the results hold for our implementation but use this simplified analysis to give a quantitative expression to the qualitative reasoning that deep MGP is scalable if its components are scalable. The analysis also allows us to compare the asymptotic performance of different approaches to parallel MGP without having to discuss which particular implementations of the basic operations can or cannot avoid certain difficult cases. We assume: (1) k is a power of two and we have unit node/edge weights, (2) $n > Cp \log p$, (3) coarsening a graph halves the number of nodes, (4) (un)coarsening or balancing a graph with n nodes takes time $\mathcal{O}(n/p + \log n)$, (5) sequential bipartitioning takes linear time. We effectively ignore edges here. This implies the assumption that nodes have bounded degree and that the degrees remain bounded when the graph is shrunk.

► **Theorem 1.** *Under the assumptions made above, deep MGP requires time*

$$\mathcal{O}\left(\frac{n}{p} \max\left(1, \log \frac{kC}{n}\right) + \log^2 n\right).$$

Proof. By (3), MGP goes through $\log(n/C)$ levels so that the overhead terms “ $\log n$ ” in (4) sum to $\mathcal{O}(\log^2 n)$ – we ignore these overhead from now on. While $> pC$ nodes are left, the graph shrinks geometrically with the levels so that the total remaining cost for (un)coarsening and balancing from (4) is linear – $\mathcal{O}(n/p)$.

When $\leq pC$ nodes are left, replication and selection of the best partition keeps the number of nodes at each level at $\Theta(pC)$. There are $\log p$ such levels incurring total cost $\mathcal{O}(C \log p)$ for (un)coarsening and balancing. By (2) this cost is bounded by $\mathcal{O}(n/p)$.

For the cost of bipartitioning we consider three cases:

Case (a) $k \leq p$: Each PE performs $\log k$ bipartitions with total cost $C \log k$. By (2) this is bounded by $\mathcal{O}(n/p)$.

Case (b) $p < k \leq n/C$: Each PE performs $\log p + k/p$ bipartitions with total cost $C(k/p + \log p)$. Once more, by (2) this is bounded by $\mathcal{O}(n/p)$.

Case (c) $k > n/C$: In this case, deep MGP first performs an n/C -way partitioning into blocks of size about C . By the above analysis, this takes time $\mathcal{O}(n/p + \log^2 n)$. Then the remaining blocks are partitioned into $k/(n/C) = kC/n$ blocks using recursive bipartitioning in time $\mathcal{O}(C \log(kC/n))$. Summing over all blocks assigned to a PE we get additional cost $n \log(kC/n)/p$. ◀

5 Implementation

In this section we describe the different components in our shared-memory parallel implementation of deep MGP called KaMinPar. Recall that the components are coarsening, bipartitioning on small graphs, and uncoarsening with k -way balancing and refinement.

5.1 Coarsening by Size-Constrained Label Propagation

We use size-constrained label propagation [34] to compute a clustering for contraction, where the weight of the heaviest cluster is restricted by a fixed upper bound U . We set $U := \varepsilon \frac{c(V)}{k'}$, where $k' = \min\{k, |V|/C\}$ is the number of blocks we obtain on the finest level (before

further bipartitioning if necessary) and ε is the imbalance from the problem formulation. This choice implies that $\frac{c(V)}{k'} + \max_v c(v) \stackrel{\max_v c(v) \leq U}{\leq} (1 + \varepsilon) \lceil \frac{c(V)}{k'} \rceil = L_{k'}$ on every level, and hence $L_{\max, k}$ simplifies to the traditional balance constraint L_k on unweighted inputs. If the current number of nodes is $\leq \frac{k'}{2}C$, we adapt k' to $\frac{k'}{2}$ and U accordingly. We perform 5 rounds of label propagation, but terminate early if no nodes were moved. To further improve the running time, we only move a node, if one of its neighbors changed its cluster in the previous round.

Parallelization. We parallelize the algorithm by iterating over all nodes in parallel. When moving a node to another cluster, we use atomic fetch-and-add operations to update the respective cluster weights. Note that we do not strictly enforce the weight limit. The limit could be violated if multiple PEs move a node to the same cluster at the same time. However, this is unproblematic in practice since the weight limit violations are usually small.

Iteration Order and Cache Locality. Solution quality of label propagation is improved when nodes are visited in increasing degree order [34, 4]. Since this is not cache efficient and lacks diversification by randomization, we sort the nodes of the graph into exponentially spaced degree buckets, i.e., bucket i contains all nodes with degree $2^i \leq d < 2^{i+1}$, and rearrange the graph such that nodes are sorted by their bucket number. For node traversal, we split buckets into small chunks and randomize node traversal on a inter-chunk and intra-chunk level. This is analogous to the randomization in Metis' matching algorithm [24].

Two-hop Clustering. We observed that size-constrained label propagation is unable to shrink some irregular graph instances sufficiently. We solve this by implementing a technique similar to the two-hop matching algorithm of Metis [31]. During label propagation, if node u cannot be moved into any neighboring cluster due to the size constraint, we store the highest rated neighboring cluster as u 's *favoured cluster*. If the graph is shrunk by less than 50% after termination, we merge singleton clusters that share the same favored cluster until the graph shrunk by 50%.

5.2 Initial Bipartitioning

We perform multilevel bipartitioning to compute an initial bipartition of a subgraph $G_{V'}$ (with $|V'| \approx 2C$). On this size, the used algorithms are sequential. For coarsening, we use label propagation and set the maximum cluster weight to the same value used in KaHiP [37]. The maximum block weight for bipartitioning is set to $L_2 := (1 + \varepsilon') \lceil \frac{c(V')}{2} \rceil$, where ε' is the adaptive imbalance as defined in Section 2. We coarsen until no further contractions are possible. For refinement, we use 2-way FM [15]. We use a pool of simple algorithms to bipartition the coarsest graph, namely random bipartitioning, breadth-first searches and greedy graph growing [24]. We repeat each algorithm several times with different random seeds and select the bipartition with the lowest edge cut. Moreover, we use the adaptive algorithm selection technique of Mt-KaHyPar [18].

5.3 Uncoarsening

After bipartitioning the blocks of a $\frac{k}{2}$ -way partition, we use a k -way balancing algorithm to restore the balance constraint (if violated). Afterwards, we run a local improvement algorithm based on size-constrained label propagation to improve it.

Balancing. In contrast to Section 4, our implementation prevents balance constraint violations by changes in $\max_v c(v)$ due to our choice of the maximum cluster weight during coarsening. However, balance violations can occur during initial bipartitioning, in particular due to our multilevel bipartitioning approach.

For each overloaded block B , we store just enough nodes of B in a priority queue P_B ordered by *relative gains*, to remove the excess weight $o(B) := c(B) - L_{\max,k}$. The *relative gain* of a node v is $d \cdot c(v)$ if $d \geq 0$ and $d/c(v)$ if $d < 0$, where d is the largest reduction in edgecut when moving v to a block that would not become overloaded. We initialize the priority queues by iterating over the nodes in G . If a node is in an overloaded block and $c(P_B) < o(B)$, we insert it. Otherwise, we only insert it if its relative gain is higher than the lowest relative gain of any element in P_B and remove its lowest element if $c(P_B) > o(B) + \max_v c(V)$ after the insertion.

Once the priority queues are initialized, we empty each overloaded block B individually by repeatedly removing the node v with the largest relative gain from P_B . If its relative gain changed since insertion, or its designated target block can no longer take v without becoming overloaded, we re-insert v (if v is still a border node). Otherwise, we move v to its target block or a random block that can take v without becoming overloaded. Subsequently, we try to insert all neighbors from its former block. To reduce the running time, we only try to insert each node once.

We parallelize the algorithm as follows. During initialization, we iterate over all nodes in parallel and maintain one thread-local priority queue for each overloaded block. Afterwards, we iterate over all blocks in parallel, merge the respective thread-local priority queues and perform node movements as described above.

Local Improvement. We use the same parallelization of size-constrained label propagation as described in Section 5.1, but strictly enforce the maximum cluster weight (set to the maximum block weight) using an atomic compare-and-swap instruction. We run at most 5 rounds of size-constrained label propagation (same value as used in Mt-KaHyPar [19]), but terminate early if no node was moved during a round.

6 Experimental Evaluation

We implemented the proposed algorithm KaMinPar in C++ and compiled it using g++-10.2 with flags `-O3 -march=native`. We use Intel’s TBB [1] as parallelization library.

Setup. We perform our experiments on two different machines. Machine A is equipped with an AMD EPYC 7702 64-Core processor clocked at 2 GHz and 1 TB main memory. This machine is only used for our scalability experiment. All other experiments are run on Machine B, which is a node of a cluster equipped with Intel Xeon Gold 6230 processors (2 sockets with 20 cores each) clocked at 2.1 GHz and 96 GB or 192 GB main memory.

We compare our algorithm with Mt-Metis 0.7.2 [30], Mt-KaHiP 1.0 [4], PuLP 0.11 [41], Metis 5.1.0 [31] and the `fsocial` preset of KaHiP 3.10 [37]. We chose this preset because it is one of the fastest configurations that computes good quality. While other presets of KaHiP achieve better partition quality, they are also much slower. We do not include ParMetis [23] and Pt-Scotch [11] in our comparison since they are slower than Mt-Metis and produce partitions with comparable solution quality [29]. Moreover, we exclude ParHiP [35] since it is outperformed by Mt-KaHiP [2]. In the following, we add a suffix to the name of each parallel partitioner to indicate the number of threads used, e.g., KaMinPar 64 for 64 threads.

Instances. We evaluate our algorithm on a benchmark set composed of 197 graphs (referred to as set A), including 129 graphs from the 10th DIMACS Implementation Challenge [6], 25 randomly generated graphs [16, 26], 25 large social networks [27, 32], and 18 graphs from various application domains [12, 44, 42]. Scalability and experiments with larger values of k are performed on a subset of set A that contains 21 graphs (referred to as set B). This benchmark set includes the 18 largest graphs (by number of nodes)² and 3 randomly chosen small graphs of set A such that a partition with 2^{20} blocks only contains a few nodes per block. Basic properties of benchmark instances are shown in Appendix A, Figure 6.

Methodology. We consider a combination of a graph and number of blocks k as an *instance*. For each instance, we usually perform several runs with different random seeds and aggregate running times and edge cuts using the arithmetic mean over all seeds. To further aggregate over multiple instances, we use the harmonic mean for relative speedups, and the geometric mean for absolute running times and edge cuts. Runs with imbalanced partitions are not excluded from aggregated running times and for instances that exceeded the time limit, we use the time limit in the aggregates. We consider an instance as infeasible, if all runs failed or computed an imbalanced partition and mark them with **X** in the plots.

To compare the solution quality of different algorithms, we use *performance profiles* [14]. Let \mathcal{A} be the set of all algorithms we want to compare, \mathcal{I} the set of instances, and $q_A(I)$ the quality of algorithm $A \in \mathcal{A}$ on instance $I \in \mathcal{I}$. For each algorithm A , we plot the fraction of instances $\frac{|\mathcal{I}_A(\tau)|}{|\mathcal{I}|}$ (y -axis) where $\mathcal{I}_A(\tau) := \{I \in \mathcal{I} \mid q_A(I) \leq \tau \cdot \min_{A' \in \mathcal{A}} q_{A'}(I)\}$ and τ is on the x -axis. Achieving higher fractions at lower τ -values is considered better. For $\tau = 1$, the y -value indicates the percentage of instances for which an algorithm performs best. Since performance profiles relate the quality of an algorithm to the best solution, the ranking induced by $\tau = 1$ does not permit full ranking of all algorithms, if more than two algorithms are included.

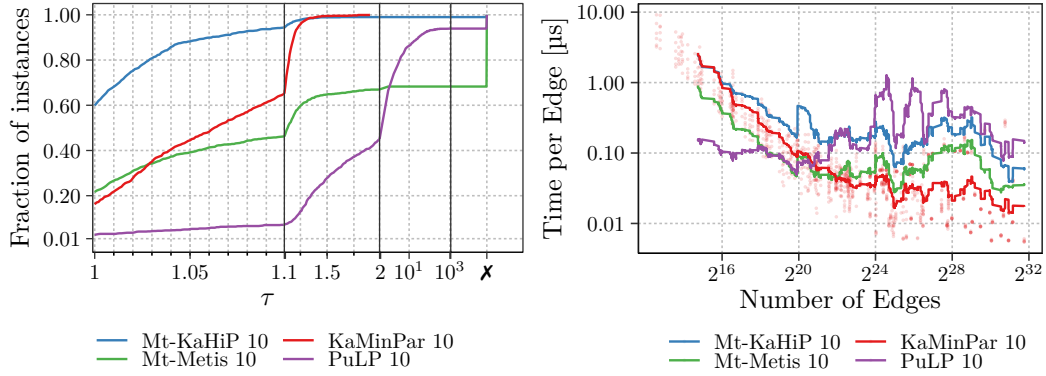
Running Time and Solution Quality for Small k . In Figure 2, Figure 3 and Table 1, we compare the quality and running time of KaMinPar with different sequential and parallel partitioners for $k \in \{2, 4, 8, 16, 32, 64\}$, $\varepsilon = 0.03$ and 5 repetitions per instance on set A and machine B. These are commonly used values to evaluate graph partitioning systems. We execute each parallel partitioner using 10 threads to simulate the performance on commodity machines.

KaMinPar 10 is the overall fastest algorithm on average and also an order of magnitude faster than the sequential partitioners Metis and KaHIP-fsocial on large graphs ($m \geq 10^8$), while producing partitions with comparable solution quality (see Figure 3 (left)). KaMinPar 10 (0.39 s geometric mean running time) is moderately faster than Mt-Metis 10 (0.48 s) and more than a factor of 2 resp. 3 faster than PuLP 10 (1.11 s) and Mt-KaHIP 10 (1.33 s). The differences in running time, as shown in Figure 2 (right), are more pronounced on larger instances, e.g., KaMinPar 10 (9.36 s) is more than factor of 3 resp. 5 faster than Mt-Metis 10 (30.36 s) resp. Mt-KaHIP 10 (55.76 s) on instances with more than 10^8 edges. Figure 2 (left) shows that Mt-KaHIP 10 computes the partition with lowest edge cut on a majority of the instances ($\approx 60\%$), while the partitions produced by PuLP 10 are more than a factor of 2 worse than the best achieved edge cuts on more than 55% of the instances. These results are expected, since Mt-KaHIP is the only partitioner that implements a parallel direct k -way FM algorithm and PuLP is the only non-multilevel system in our evaluation.

² excluding **er-fact1.5-scale26**, since Mt-KaHiP and Mt-Metis are unable to compute a partition on this graph even for small k , and **kmer_V2a** to avoid over-representation of k -mer graphs

■ **Table 1** Geometric mean running time and solution quality for different algorithms on benchmark set A and $k \in \{2, 4, 8, 16, 32, 64\}$. Running time only includes instances for which all algorithms produced a result. The number of included instances is shown in the last row. Solution quality is relative to KaMinPar (lower is better) and only includes instances for which the respective algorithm computed a balanced partition. Thus, solution quality cannot be compared between different competitors.

Algorithm	T	$T[m \geq 10^6]$	$T[m \geq 10^8]$	rel. cut	# infeasible
KaMinPar 10	0.39 s	0.85 s	9.36 s	1.00	0
Mt-Metis 10	0.48 s	1.49 s	30.36 s	1.00	349
Mt-KaHiP 10	1.33 s	3.84 s	55.76 s	0.94	6
PuLP 10	1.11 s	5.70 s	95.93 s	2.39	72
Metis	1.00 s	4.15 s	97.44 s	1.05	2
KaHiP-fsocial	2.93 s	11.05 s	200.67 s	1.03	8
# instances	1,150	832	196		



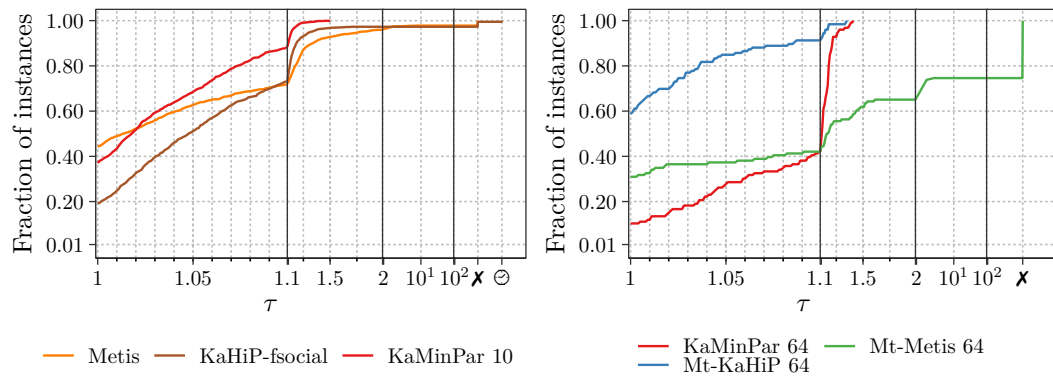
■ **Figure 2** Performance profile and running time plot (shows time per edge with a right-aligned rolling geometric mean over 50 instances) comparing the performance of KaMinPar with different partitioners for $k \in \{2, 4, 8, 16, 32, 64\}$ on set A.

We also ran KaMinPar, Mt-Metis and Mt-KaHiP on all 64 cores of machine A. Here, we partitioned each graph of the reduced benchmark set B into $k \in \{2, 4, 8, 16, 32, 64\}$ blocks allowing a maximum imbalance of $\varepsilon = 0.03$. In this setup, KaMinPar 64 is 5 resp. 4.4 times faster than Mt-KaHiP 64 resp. Mt-Metis 64, whereas on the same instances on 10 cores of machine B, it is 5.7 resp. 3.1 times faster. Thus, we can see that the running time of KaMinPar scales slightly worse to 64 cores than Mt-KaHiP, but slightly better than Mt-Metis for smaller values of k . In Figure 3 (right), we compare the solution quality of each algorithm on the reduced benchmark set B. We can see that the differences of the partitioners in terms of solution quality are more pronounced on this set than on set A (compare with Figure 2). However, this is due to the benchmark set, since each partitioner produced partitions with comparable quality if we compare them individually with 10 and 64 threads. Overall, we can conclude that KaMinPar offers a compelling trade-off between running time and quality compared to established shared-memory and sequential GP systems.

Running Time and Solution Quality for Large k . In Table 2, we present the results of our experiment with different parallel partitioners (each using 10 threads) for larger values of $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$, $\varepsilon = 0.03$ and 3 repetitions per instance with a time limit of one

■ **Table 2** Results of our experiment for large values of k with different parallel partitioners on set B. The last two columns show the geometric mean running time and edge cuts relative to KaMinPar of all instances that do not crash (timeout instances are additionally excluded in edge cut comparison).

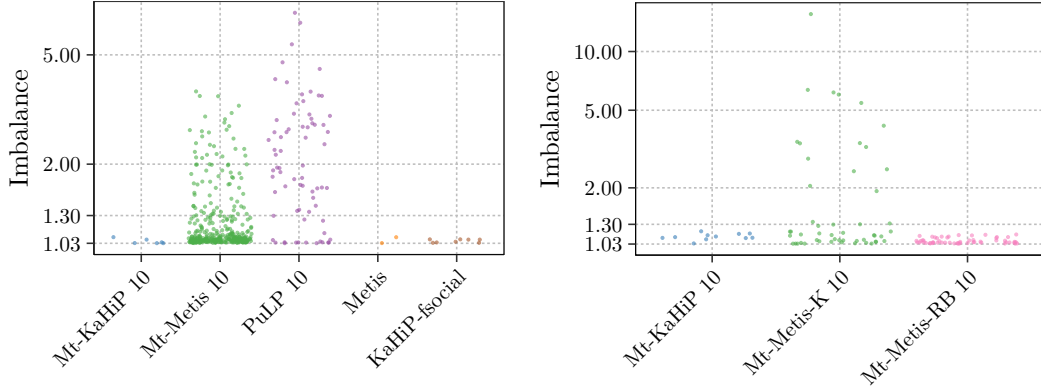
Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar 10	0	0	0	84	1.00	1.00
Mt-Metis-K 10	19	10	51	4	11.91	0.99
Mt-Metis-RB 10	0	25	55	4	5.61	1.03
Mt-KaHiP 10	31	7	11	35	38.64	1.00
PuLP 10	76	0	0	8	73.52	1.25



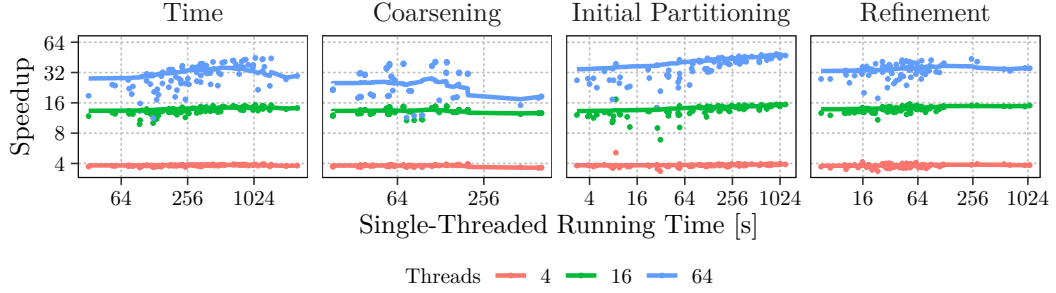
■ **Figure 3** Left: performance profile of KaMinPar, Metis and KaHiP-fsocial on benchmark set A with $k \in \{2, 4, 8, 16, 32, 64\}$ and $\varepsilon = 0.03$. Right: performance profile of KaMinPar, Mt-KaHiP and Mt-Metis on 64 cores of machine A, reduced benchmark set B with $k \in \{2, 4, 8, 16, 32, 64\}$ and $\varepsilon = 0.03$. Note that the change in relative solution quality is due to the reduced benchmark set.

hour on set B and machine B (192 Gb main memory). Note that the time limit is 10 times larger than the longest running time of KaMinPar for an instance. We additionally included the recursive bipartitioning version of Mt-Metis (referred to as Mt-Metis-RB) to evaluate the performance of recursive bipartitioning on large k . In the following, we consider a run of an algorithm for an instance as *feasible*, if the algorithm terminates in the given time limit and the produced partition satisfies the balance constraint $L_k := (1 + \varepsilon) \lceil \frac{c(V)}{k} \rceil$.

Out of the 84 evaluated instances (21 graphs times 4 values of k), Mt-Metis-RB 10, Mt-Metis-K 10, PuLP 10 and Mt-KaHiP 10 were only able to produce on 4, 4, 8 resp. 35 instances a feasible solution. Mt-Metis-RB 10 and Mt-Metis-K 10 primarily failed to produce solutions that satisfy the balance constraint (55 resp. 51 instances). Figure 4 (right) shows that Mt-Metis-K 10 generally produces larger balance violations (median resp. maximum imbalance is 1.14 resp. 15.56) than Mt-Metis-RB 10 (median 1.05 and maximum 1.15). Mt-KaHiP 10 and PuLP 10 were mostly unable to compute a partition in the given time limit (31 resp. 76 instances). KaMinPar 10 produced a feasible solution on all instances. The fastest competitor is Mt-Metis-RB 10, which is more than 5 times slower than KaMinPar 10 on average. All other partitioners are an order of magnitude slower. If we include all imbalanced partitions and individually compare the partitioners on those instances with respect to solution quality, we can see that all perform comparable (except for PuLP 10). However, a fair comparison is difficult due to the large number of infeasible solutions. We can conclude that KaMinPar is currently the only partitioner considered in our evaluation that can reliably compute feasible partitions for larger values of k in a reasonable amount of time.



■ **Figure 4** Left: imbalance of infeasible partitions computed by Mt-KaHiP 10, Mt-Metis 10, PuLP 10, Metis and KaHiP-fsocial on benchmark set A with $k \in \{2, 4, 8, 16, 32, 64\}$ and $\varepsilon = 0.03$. Right: imbalance of infeasible partitions computed by Mt-Metis-K 10, Mt-Metis-RB 10 and Mt-KaHiP 10 on benchmark set B with $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$ and $\varepsilon = 0.03$.



■ **Figure 5** Self-relative speedups for the different components of KaMinPar on set B.

Scalability of KaMinPar. In Figure 5, we show the scalability of KaMinPar for $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$, $\varepsilon = 0.03$ and three repetitions per instance on set B using $p \in \{1, 4, 16, 64\}$ cores of machine A. In the plot, we represent the speedup of each instance as a point and the cumulative harmonic mean speedup over all instances with a single-threaded running time $\geq x$ seconds with a line. Note that initial partitioning includes all calls to our initial bipartitioning algorithms on graphs with more than $2pC$ nodes.

The overall harmonic mean speedup of KaMinPar is 3.8 for $p = 4$, 13.3 for $p = 16$ and 27.9 for $p = 64$. The harmonic mean speedups of coarsening, initial partitioning and refinement are 25.0, 34.5 and 33.1 for $p = 64$. We note that our refinement component achieves slightly better speedups than our coarsening component, although both are based on the size-constrained label propagation algorithm. This effect is most pronounced on instances with larger node degrees. During coarsening, each thread aggregates ratings to neighboring clusters in a local hash map (with only 2^{15} entries) for nodes with small degree and in a local vector of size n for high degree nodes ($\geq 2^{15}/3$). During refinement, each thread uses a local vector of size k for this. Instances with larger node degrees more often uses the local vector of size n to aggregate ratings during coarsening, which can limit scalability due to cache effects. Note that KaMinPar performs no expensive arithmetic operations. Hence, perfect speedups are not possible due to limited memory bandwidth.

7 Conclusion and Future Work

We presented a new graph partitioning scheme that successfully combines the merits of classical direct k -way partitioning and recursive bipartitioning. Similar to direct k -way partitioning, deep MGP coarsens and uncoarsens the graph only once, and allows the use of k -way local improvement algorithms. Yet, it does not suffer scalability problems if k is large and has a better asymptotic running time than recursive bipartitioning. Our experimental evaluation shows that our shared-memory parallel implementation of deep MGP runs efficiently on up to 64 PEs, while achieving comparable results to established graph partitioners if k is small. Furthermore, our evaluation showed that KaMinPar is an order of magnitude faster than other graph partitioners based on direct k -way partitioning if k is large, while consistently producing balanced solutions. In the future, we would like to explore graph partitioning for very large values of k , e.g., $k \in \Theta(n)$.

References

- 1 Intel Threading Building Blocks. <https://www.threadingbuildingblocks.org/>.
- 2 Y. Akhremtsev. *Parallel and External High Quality Graph Partitioning*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2019. doi:10.5445/IR/1000098895.
- 3 Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag. Engineering a direct k -way Hypergraph Partitioning Algorithm. In *19th Workshop on Algorithm Engineering and Experiments, (ALENEX)*, pages 28–42, 2017.
- 4 Y. Akhremtsev, P. Sanders, and C. Schulz. High-Quality Shared-Memory Graph Partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2710–2722, 2020. doi:10.1109/TPDS.2020.3001645.
- 5 K. Andreev and H. Räcke. Balanced Graph Partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
- 6 D. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *10th DIMACS Implementation Challenge – Graph Partitioning and Graph Clustering*, 2012.
- 7 C. Bichot and P. Siarry, editors. *Graph Partitioning*. Wiley, 2011.
- 8 A. Buluc, H. Meyerhenke, I. A. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering*, volume 9220 of *LNCS*, pages 117–158. Springer, 2014.
- 9 U. V. Catalyürek and C. Aykanat. Hypergraph-partitioning based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 10(7):673–693, 1999.
- 10 Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Ucar. Multithreaded Clustering for Multi-Level Hypergraph Partitioning. In *IEEE 26th International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 848–859. IEEE, 2012.
- 11 C. Chevalier and F. Pellegrini. PT-Scotch. *Parallel Computing*, pages 318–331, 2008.
- 12 T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.
- 13 K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and Ü. V. Catalyürek. Parallel Hypergraph Partitioning for Scientific Computing. In *20th International Conference on Parallel and Distributed Processing, (IPDPS)*, pages 124–124. IEEE, 2006.
- 14 E. D. Dolan and J. J. Moré. Benchmarking Optimization Software with Performance Profiles. *Math. Program.*, 91(2):201–213, 2002. doi:10.1007/s101070100263.
- 15 C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of the 19th Conference on Design Automation*, pages 175–181, 1982.
- 16 D. Funke, S. Lamm, P. Sanders, C. Schulz, D. Strash, and M. von Looz. Communication-free Massively Distributed Graph Generation. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.

- 17 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W.H. Freeman, San Francisco, 1979.
- 18 L. Gottesbüren, T. Heuer, P. Sanders, and S. Schlag. Shared-Memory n-level Hypergraph Partitioning. *arXiv preprint arXiv:2104.08107*, 2021.
- 19 L. Gottesbüren, T. Heuer, P. Sanders, and S. Schlag. Scalable Shared-Memory Hypergraph Partitioning. In *23rd Workshop on Algorithm Engineering and Experiments, (ALENEX 2021)*, pages 16–30. SIAM, 2021.
- 20 T. Heuer, N. Maas, and S. Schlag. Multilevel Hypergraph Partitioning with Vertex Weights Revisited. In *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*, volume 190 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SEA.2021.8.
- 21 T. Heuer and S. Schlag. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In *16th International Symposium on Experimental Algorithms, (SEA)*, pages 21:1–21:19, 2017.
- 22 M. Holtgrewe, P. Sanders, and C. Schulz. Engineering a Scalable High Quality Graph Partitioner. *24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12, 2010.
- 23 G. Karypis and V. Kumar. Parallel Multilevel k -way Partitioning Scheme for Irregular Graphs. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 1996.
- 24 G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
- 25 G. Karypis and V. Kumar. Multilevel k -way Partitioning Scheme for Irregular Graphs. *Journal on Parallel and Distributed Computing*, 48(1):96–129, 1998.
- 26 F. Khorasani, R. Gupta, and L. N. Bhuyan. Scalable SIMD-Efficient Graph Processing on GPUs. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques, PACT '15*, pages 39–50, 2015.
- 27 University of Milano Laboratory of Web Algorithms. Datasets. URL: <http://law.di.unimi.it/datasets.php>.
- 28 D. LaSalle and G. Karypis. Multi-threaded Graph Partitioning. In *27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 225–236, 2013.
- 29 D. Lasalle and G. Karypis. Multi-threaded Graph Partitioning. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, pages 225–236, 2013. doi:10.1109/IPDPS.2013.50.
- 30 D. LaSalle and G. Karypis. A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning. In *45th International Conference on Parallel Processing (ICPP)*, pages 236–241, 2016.
- 31 D. LaSalle, Md. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis. Improving Graph Partitioning for Modern Graphs and Architectures. In *Proceedings of the 5th Workshop on Irregular Applications - Architectures and Algorithms, IA3 2015, Austin, Texas, USA, November 15, 2015*, pages 14:1–14:4. ACM, 2015. doi:10.1145/2833179.2833188.
- 32 J. Leskovec. Stanford Network Analysis Package (SNAP).
- 33 M. Birn and V. Osipov and P. Sanders and C. Schulz and N. Sitchinava. Efficient Parallel and External Matching. In *Euro-Par*, volume 8097 of *LNCS*, pages 659–670. Springer, 2013.
- 34 H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning Complex Networks via Size-Constrained Clustering. In *Experimental Algorithms*, volume 8504 of *LNCS*, pages 351–363. Springer, 2014. doi:10.1007/978-3-319-07959-2_30.
- 35 H. Meyerhenke, P. Sanders, and C. Schulz. Parallel Graph Partitioning for Complex Networks. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2625–2638, 2017. doi:10.1109/TPDS.2017.2671868.
- 36 U. N. Raghavan, R. Albert, and S. Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical review E*, 76(3):036106, 2007.

- 37 P. Sanders and C. Schulz. Engineering Multilevel Graph Partitioning Algorithms. In *19th European Symposium on Algorithms (ESA)*, volume 6942 of *LNCS*, pages 469–480. Springer, 2011.
- 38 S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. k -way Hypergraph Partitioning via n -Level Recursive Bisection. In *18th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67, 2016.
- 39 K. Schloegel, G. Karypis, and V. Kumar. Graph Partitioning for High Performance Scientific Simulations. In *The Sourcebook of Parallel Computing*, pages 491–541, 2003.
- 40 C. Schulz and D. Strash. Graph Partitioning: Formulations and Applications to Big Data. In *Encyclopedia of Big Data Technologies*. Springer, 2019. doi:10.1007/978-3-319-63962-8_312-2.
- 41 G. M. Slota, K. Madduri, and S. Rajamanickam. PuLP: Scalable Multi-Objective Multi-Constraint Partitioning for Small-World Networks. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 481–490, 2014. doi:10.1109/BigData.2014.7004265.
- 42 N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The DAC 2012 Routability-driven Placement Contest and Benchmark Suite. In *49th Design Automation Conference, (DAC)*, pages 774–782, 2012.
- 43 C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview. In *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pages 27–58. 2007.
- 44 S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2007. doi:10.1145/1362622.1362674.

A Benchmark Set Statistics

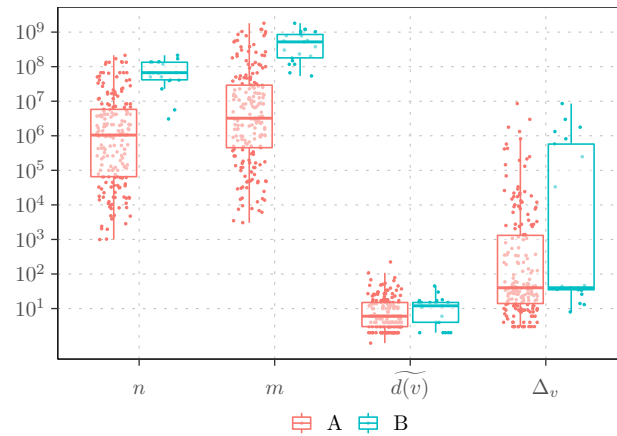


Figure 6 Basic properties of benchmark sets A and B: number of nodes n , number of edges m , median degree $\widehat{d(v)}$ and maximum degree Δ_v .

Faster $(1 + \epsilon)$ -Approximation for Unsplittable Flow on a Path via Resource Augmentation and Back

Fabrizio Grandoni ✉

IDSIA, USI-SUPSI, Lugano, Switzerland

Tobias Mömke ✉ 

Department of Computer Science, Universität Augsburg, Germany

Andreas Wiese ✉

Department of Industrial Engineering, Universidad de Chile, Santiago, Chile

Abstract

Unsplittable flow on a path (UFP) is an important and well-studied problem. We are given a path with capacities on its edges, and a set of tasks where for each task we are given a demand, a subpath, and a weight. The goal is to select the set of tasks of maximum total weight whose total demands do not exceed the capacity on any edge. UFP admits an $(1 + \epsilon)$ -approximation with a running time of $n^{O_\epsilon(\text{poly}(\log n))}$, i.e., a QPTAS [Bansal et al., STOC 2006; Batra et al., SODA 2015] and it is considered an important open problem to construct a PTAS. To this end, in a series of papers polynomial time approximation algorithms have been developed, which culminated in a $(5/3 + \epsilon)$ -approximation [Grandoni et al., STOC 2018] and very recently an approximation ratio of $(1 + \frac{1}{e+1} + \epsilon) < 1.269$ [Grandoni et al., 2020]. In this paper, we address the search for a PTAS from a different angle: we present a faster $(1 + \epsilon)$ -approximation with a running time of only $n^{O_\epsilon(\log \log n)}$. We first give such a result in the relaxed setting of resource augmentation and then transform it to an algorithm *without* resource augmentation. For this, we present a framework which transforms algorithms for (a slight generalization of) UFP under resource augmentation in a black-box manner into algorithms for UFP *without* resource augmentation, with only negligible loss.

2012 ACM Subject Classification Theory of computation \rightarrow Dynamic programming; Theory of computation \rightarrow Packing and covering problems; Theory of computation \rightarrow Rounding techniques

Keywords and phrases Approximation Algorithms, Unsplittable Flow, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.49

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Excellence Grant 200020B_182865/1. *Tobias Mömke*: Partially supported by the DFG Grant 439522729 (Heisenberg-Grant), the ERC Advanced Investigators Grant 695614 (POWVER), and by DFG Grant 389792660 as part of TRR 248 (CPEC).

Andreas Wiese: Partially supported by FONDECYT Regular grants 1170223 and 1200173.

1 Introduction

The unsplittable flow on a path problem (UFP) is a natural packing problem which has received a lot of attention, e.g., [3, 17, 15, 6, 7, 4, 8, 9]. We are given a path $G = (V, E)$ with a capacity $u(e) \in \mathbb{N}_0$ for each edge $e \in E$ and a set of tasks T . For each task $i \in T$ we are given a sub-path $P(i)$ of E , a demand $d(i) \in \mathbb{N}_0$, and a weight (or profit) $w(i) \in \mathbb{N}_0$. For any set of tasks $T' \subseteq T$, we define $d(T') := \sum_{i \in T'} d(i)$ and $w(T') := \sum_{i \in T'} w(i)$ and for each $e \in E$ we define $T_e \subseteq T$ to be the set of all tasks $i \in T$ using e , i.e., such that $e \in P(i)$. Similarly we sometimes say that i is contained, contains, or intersects a subpath P if $P(i)$ does. The goal is to select a set of tasks $T' \subseteq T$ of maximum total weight $w(T')$ such that T' obeys the edge capacities, i.e., $d(T' \cap T_e) \leq u(e)$ for each edge $e \in E$. We denote by n the size of the input, and hence in particular $|T| \leq n$ and $|E| \leq n$.



© Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 49; pp. 49:1–49:15

Leibniz International Proceedings in Informatics



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

UFP is a generalization of KNAPSACK (i.e., if $|E| = 1$) and it is motivated by various applications. For example, the path G can represent a network with a chain of communication links in which we seek to select the most profitable set of possible transmissions that obey the given edge capacities. Also, the edges in E can correspond to discrete time slots, each task models a job that we might want to execute, and the edge capacities model the available amount of a resource shared by the jobs like energy or machines. Also, there is a connection between UFP and general caching, i.e., where pages can have different (possible non-unit) sizes and different costs for being evicted [12].

There is a $(1 + \epsilon)$ -approximation algorithm known for UFP with running time of $n^{O_\epsilon(\log n)}$ [7], i.e., a QPTAS (improving an earlier QPTAS in [5]), and it has been a long-standing open problem to construct a PTAS. Towards this goal, polynomial time approximation algorithms for UFP have been developed and the best known approximation ratio has been gradually improved from $O(\log n)$ [6] to $7 + \epsilon$ [8], $2 + \epsilon$ [3], and $5/3 + \epsilon$ [17], while the currently best known ratio is $1 + \frac{1}{1+\epsilon} + \epsilon < 1.269$ [15].

1.1 Our Contribution

In this paper, we contribute to the search for a PTAS for UFP from a different angle: we improve the *running time* of the known QPTAS for UFP [7] and present a $(1 + \epsilon)$ -approximation with a running time of only $n^{O_\epsilon(\log \log n)}$. Hence, our result is in the same spirit as similar improvements for the Maximum Independent Set of Rectangles problem in [13] and precedence constrained scheduling for unit-size jobs in [18].

We first present our result for the case of resource augmentation, i.e., when we allow ourselves to increase the edge capacities by a factor of $1 + \delta$ for some constant $\delta > 0$ while the compared optimal solution OPT does not have this privilege. Let u_{\min} and u_{\max} denote the minimum and maximum edge capacities, respectively. In that setting, it is easy to show that we can assume that the edge capacities are in a constant range, namely $u_{\max} = O_{\epsilon, \delta}(u_{\min})$. We classify each task $i \in T$ to be *small* or *large*, depending on whether i uses a relatively small or a relatively large fraction of the available capacity on the edges of $P(i)$. Since we have a constant range of edge capacities, one can show easily that each edge e can be used by only a constant number of large tasks in OPT .

The high level strategy in the known QPTASs [5, 7] is to take the middle edge e^* , guess the large tasks using e^* , split the small tasks using e^* into $O(\log n)$ or $(\log n)^{O(1)}$ groups, and guess an *under-estimating capacity profile* for each group with $O_\epsilon(1)$ uniform steps. In more detail, in the latter step one guesses for each edge $e \in E$ approximately how much capacity from $u(e)$ is used in OPT by the tasks crossing e^* in each group. For each group one argues that one loses only a factor of $1 + \epsilon$ in the profit by underestimating the true capacity. Then one recurses on the subpaths of E on the left of e^* and on the right of e^* which yields a recursion depth of $O(\log n)$.

Instead, when we consider the small tasks using e^* , we use only *one over-estimating* profile with *non-uniform* step size for *all* tasks together with only $O_{\epsilon, \delta}(\log \log n)$ steps. Each step is a power of $1 + \delta$ in $[\frac{\delta}{\log n} u_{\min}, u_{\max}]$. This yields $O_{\epsilon, \delta}(\log \log n)$ steps since the profile is monotone on the left and on the right of e^* and $u_{\max} = O_{\epsilon, \delta}(u_{\min})$. Also, our justification for the error is very different. On some edges e the error is at most $\frac{\delta}{\log n} u_{\min}$ which accumulates to at most $O(\delta)u_{\min} \leq O(\delta)u(e)$ during the recursion. On the other edges e the error is at most a δ -fraction of the total demand used on e by tasks crossing e^* ; over the recursion this can add up to at most $\delta u(e)$ since in OPT edge e is used by tasks with a total demand of at most $u(e)$.

When we recurse we employ another novelty: we consider the tasks crossing at least one of *two* specially defined edges e_1^*, e_2^* , and recurse in *three* subpaths induced by them: e_1^* is like before the middle edge of current subpath. The other edge e_2^* is chosen such that half of previously guessed steps are on the left and the other half on the right (like in the QPTAS in [7]). This ensures that *at the same time* the recursion depth is $O(\log n)$ and each recursive call is described by a subpath of E and only $O_{\epsilon, \delta}(\log \log n)$ steps from previously guessed profiles. Thus, we can embed this recursion into a dynamic program (DP) with DP-table of size $n^{O_{\epsilon, \delta}(\log \log n)}$.

Then, we present a new framework using which one can transform algorithms for UFP under resource augmentation (like ours) in a black-box manner into algorithms for UFP *without* resource augmentation, while losing only a factor of $1 + \epsilon$ in the approximation ratio. To this end, we define a slight generalization of UFP that we denote by *Bonus-UFP*. The key difference to UFP is that in addition to profit from normal tasks, one receives a bonus for subpaths which do not completely contain the path of any selected task. Also, its instances are required to have a simpler structure than general UFP instances, yielding similar properties as obtained via resource augmentation. For example, on each edge one is allowed to select only a constant number of large tasks (independently of the actual edge capacities!), so they can be easily guessed in time $n^{O_{\epsilon}(1)}$ for each edge. Also, the capacity allocated for small tasks in *OPT* is intuitively in a constant range (similarly as above) and the notion of resource augmentation is defined such that this capacity for the small tasks is increased by a factor $1 + \delta$.

Our framework directly transforms any algorithm for Bonus-UFP under resource augmentation to an algorithm for UFP *without* resource augmentation, while increasing the approximation ratio only by a factor $1 + \epsilon$. The transformation uses the slack-lemma [16] which was employed in previous algorithms for UFP [16, 15, 17] in order to gain free capacity on the edges within quite complicated dynamic programs. With our framework one does not need to apply the slack-lemma and construct this technical machinery “by hand” anymore, but it is sufficient to design an algorithm for Bonus-UFP under resource augmentation and then the framework does the transformation automatically.

► **Theorem 1** (informal). *Given an α -approximation algorithm for Bonus-UFP under resource augmentation with a running time of $T(n)$, we can construct a $(1 + \epsilon)\alpha$ -approximation algorithm for UFP (without resource augmentation) with a running time of $T(n)n^{O_{\epsilon}(1)}$.*

We hope that this facilitates future research on UFP for (eventually) finding a PTAS. In particular, we believe that if one constructs an algorithms for UFP under resource augmentation then it is very likely that it can be adjusted to an algorithm for Bonus-UFP under resource augmentation. For example, we demonstrate that this can be easily done with our algorithm for UFP under resource augmentation above, which yields the following theorem.

► **Theorem 2.** *For any $\epsilon > 0$ there is a $(1 + \epsilon)$ -approximation algorithm for UFP with a running time of $n^{O_{\epsilon}(\log \log n)}$.*

1.2 Other related work

There are some special cases of UFP for which PTASs are known, for example when there are $O(1)$ edges such that each input task uses one of them [16], each input task can be selected an unbounded number of times [16], or when the profit of each task is proportional to its demand [7]. Also, there is an FPT- $(1 + \epsilon)$ -algorithm known for the unweighted case of UFP where the fixed parameter is the cardinality of the optimal solution [19].

Variations of UFP have been studied like bagUFP where the input tasks are partitioned into bags, and we are allowed to select at most one task from each bag [14]. Also, since the natural LP-formulation of UFP suffers from an integrality gap of $\Omega(n)$ [9], stronger LP-formulations have been investigated [10, 2]. Furthermore, unsplittable flow has been studied on trees, where the best known results are a $O(\log^2 n)$ -approximation [10]. This was generalized to a $O(k \cdot \log n)$ -approximation [1] for submodular objectives where k denotes the pathwidth of the given tree (bounded by $O(\log n)$).

2 Faster approximation scheme for UFP with resource augmentation

We first provide a $(1+\epsilon)$ -approximation algorithm for UFP with a running time of $n^{O_{\epsilon,\delta}(\log \log n)}$ for the simplified setting of δ -resource augmentation where we permit the algorithm to compute a solution S where for each edge $e \in E$ we have that $\sum_{i \in S \cap T_e} d(i) \leq (1 + \delta)u(e)$ and we require the optimal solution OPT to respect the original capacities $u(\cdot)$, i.e., $\sum_{i \in \text{OPT} \cap T_e} d(i) \leq u(e)$ for each $e \in E$. For a UFP instance with a path $G = (V, E)$ and edge capacities $u(\cdot)$, we define $u_{\min} := \min_{e \in E} u(e)$ and $u_{\max} := \max_{e \in E} u(e)$. First, we use the resource augmentation to reduce the general case to the setting of a constant range of (polynomially bounded) edge capacities where $u_{\max} \leq u_{\min}/\delta^{(2/\epsilon)}$.

► **Lemma 3.** *Assume that for any constants $\epsilon > 0$, $\delta > 0$, there is an α -approximation algorithm under δ -resource augmentation for UFP with a running time of $T(n)$ for instances such that $u_{\min} = n/\delta$ and $u_{\max} \leq u_{\min}/\delta^{2/\epsilon}$. Then there is an $(\alpha+\epsilon)$ -approximation algorithm for UFP under 4δ -resource augmentation with a running time of $T(n)(n \log u_{\max})^{O_{\epsilon,\delta}(1)}$.*

Due to Lemma 3, in the following we assume that we are given an instance of UFP such that $n/\delta \leq u(e) \leq n/(\eta\delta)$ for each $e \in E$ and some $\epsilon, \delta > 0$, with $\eta := \delta^{2/\epsilon}$, and that we are in the setting of δ -resource augmentation. By contracting edges suitably we can assume w.l.o.g. that each edge $e \in E$ is the first or the last edge of the path $P(i)$ of some input task $i \in T$.

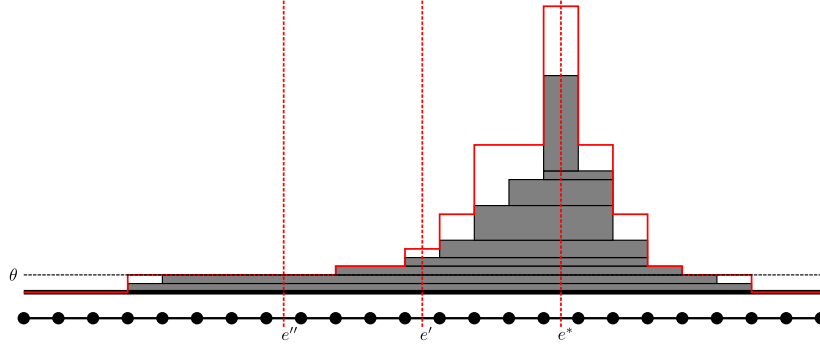
2.1 Recursive Algorithm

We describe our algorithm first as a recursive algorithm and then embed it into a dynamic program which will have a running time of $n^{O_{\epsilon,\delta}(\log \log n)}$.

Let $e^* \in E$ be an edge in the middle of E , i.e., such that at most $\lfloor |E|/2 \rfloor$ edges are on the left of e^* and at most $\lfloor |E|/2 \rfloor$ edges are on the right of e^* . We would like to guess the tasks in $\text{OPT} \cap T_{e^*}$. Since there are too many possibilities for this, we first guess approximately how much capacity the tasks in $\text{OPT} \cap T_{e^*}$ use on each edge $e \in E$. For any set of tasks $S \subseteq T$, we define $u_S(e) := \sum_{i \in S \cap T_e} d(i)$ for each $e \in E$. Observe that $u_{T_{e^*} \cap \text{OPT}}$ is non-decreasing on the left of e^* and non-increasing on the right of e^* , since all tasks in $T_{e^*} \cap \text{OPT}$ use e^* . We will guess an *over-estimating* profile $\text{mp}_{T_{e^*} \cap \text{OPT}}: E \rightarrow \mathbb{N}$ of $u_{T_{e^*} \cap \text{OPT}}(e)$ defined below, with the properties that

- for each edge e it holds that $\text{mp}_{T_{e^*} \cap \text{OPT}}(e) \geq u_{T_{e^*} \cap \text{OPT}}(e)$,
- $\text{mp}_{T_{e^*} \cap \text{OPT}}$ is a step-function with only $O_{\epsilon,\delta}(\log \log n)$ steps (i.e., E can be partitioned into $O_{\epsilon,\delta}(\log \log n)$ subpaths and $\text{mp}_{T_{e^*} \cap \text{OPT}}$ is constant on each subpath), and
- $\text{mp}_{T_{e^*} \cap \text{OPT}}(e)$ and $u_{T_{e^*} \cap \text{OPT}}(e)$ do not differ too much on each edge $e \in E$.

Formally, let $D := \log n$; our overall algorithm will intuitively be a recursion with D levels. For any set of tasks $S \subseteq T$ we define the *minimal profile* $\text{mp}_S: E \rightarrow \mathbb{N}$ of S as follows. Let $\theta := \lfloor u_{\min} \cdot \delta/D \rfloor$. If for an edge $e \in E$ it holds that $d(S \cap T_e) = 0$ then we define



■ **Figure 1** Overestimating profile $\text{mp}(u_{T_{e^*} \cap \text{OPT}})$ (the red line). The edge e' halves the number of steps and e'' the length of the path on the left-hand side of e^* .

$\text{mp}_S(e) := 0$. Otherwise, we define $\text{mp}_S(e) := \lfloor \theta(1 + \delta)^{k_e} \rfloor$ where k_e is the smallest integers such that $\theta(1 + \delta)^{k_e} \geq d(S \cap T_e)$ (see Figure 1). It turns out that $\text{mp}_S(\cdot)$ is a step-function with only $O_{\delta, \epsilon}(\log \log n)$ steps, assuming that there is an edge \bar{e} that is used by all tasks in S (like the edge e^* above).

► **Lemma 4.** *There is a value $\Gamma = O_{\delta, \epsilon}(\log D) = O_{\delta, \epsilon}(\log \log n)$ such that for every edge $\bar{e} \in E$ and every set $S \subseteq T_{\bar{e}}$ with $d(S \cap T_e) \leq u(e)$ for each $e \in E$, we have that $\text{mp}_S(\cdot)$ is a step-function with at most Γ steps.*

Proof. Recall that by Lemma 3 $u_{\min} = n/\delta$ and $u_{\max} \leq n/(\delta\eta)$. We therefore search for an upper bound on the smallest value k such that $\lfloor \theta(1 + \delta)^k \rfloor \geq n/(\delta\eta)$. We have that

$$\begin{aligned} k &\in O(\log_{1+\delta}\left(\frac{n}{\delta\eta\theta}\right)) = O\left(\log_{1+\delta}\left(\frac{nD}{\eta\delta^2 u_{\min}}\right)\right) \\ &= O\left(\log_{1+\delta}\left(\frac{D}{\eta\delta^2}\right)\right) = O_{\delta, \epsilon}(\log D) = O_{\delta, \epsilon}(\log \log n). \end{aligned}$$

The second claim follows due to monotonicity of the profile formed by the tasks $\text{OPT}_S \cap T_e$. ◀

For any set S , the profile $\text{mp}_S(\cdot)$ overestimates the true demand of the tasks S on each edge e . Therefore, we define the error of this estimation by $\text{err}(\text{mp}_S, e) := \text{mp}_S(e) - u_S(e)$ for each edge $e \in E$. Intuitively, in our algorithm we will guess profiles mp_S in each of the $D = \log n$ recursion levels, such that at the end each edge e is in the support of at most $2D$ of these profiles. A key insight is the following lemma which implies that for each edge e , the sum of the errors of these $2D$ profiles is bounded by the extra space of $\delta u(e)$ due to the resource augmentation. Intuitively, if $\text{mp}_S(e) = \theta$ on some edge e , then the error is at most $\theta = u_{\min} \cdot \delta/D$ and for $2D$ profiles the errors of this type can accumulate to at most $2D \cdot \theta \leq 2\delta u_{\min} \leq 2\delta u(e)$. On the other hand, if $\text{mp}_S(e) > \theta$ then the error is at most $\delta \cdot d(S \cap T_e)$, and this can accumulate to at most $\delta \cdot u(e)$ for all profiles together if these profiles correspond to disjoint sets of tasks $S_1, S_2, \dots, S_{D'}$ that use at most $u(e)$ units of capacity on e altogether, for any D' .

► **Lemma 5.** *Let $S_1, S_2, \dots, S_{D'} \subseteq T$ be disjoint sets of tasks with $D' \in O(D)$. If $\sum_j d(S_j \cap T_e) \leq u(e)$ for an edge e , then $\sum_{j=1}^{D'} \text{err}(\text{mp}_{S_j \cap T_e}, e) \leq O(\delta) \cdot u(e)$.*

Proof. For each index j and edge e such that $\text{mp}_{S_j \cap T_e}(e) = \lfloor \theta(1+\delta)^{k_j} \rfloor$ for some k_j , the error $\text{err}(\text{mp}_{S_j \cap T_e}, e)$ is bounded from above by $\lfloor \theta \rfloor$ if $k_j = 0$ and by $\lfloor \theta \cdot (1+\delta)^{k_j} \rfloor - \lceil \theta \cdot (1+\delta)^{k_j-1} \rceil \leq \theta(1+\delta)^{k_j-1} \cdot \delta \leq \delta d(S_j)$ if $k_j > 0$. (Observe that the rounding is valid because the demands are integers.) Thus $\text{err}(\text{mp}_{S_j \cap T_e}, e) \leq \max\{\theta, \delta d(S_j)\}$. Summing up over all errors we obtain

$$\sum_{j=1}^{D'} \text{err}(\text{mp}_{S_j \cap T_e}, e) \leq D' \cdot \theta + \sum_j \delta d(S_j \cap T_e) \leq \frac{D' u_{\min} \delta}{D} + \delta u(e) \in O(\delta)u(e). \quad \blacktriangleleft$$

As mentioned above, we guess $\text{mp}_{T_{e^*} \cap \text{OPT}}$, which can be done in time $n^{O_{\delta, \epsilon}(\log \log n)}$ due to Lemma 4. Then, we compute the essentially most profitable set of tasks $T' \subseteq T_{e^*}$ that fits into $\text{mp}_{T_{e^*} \cap \text{OPT}}$, i.e., such that $d(T' \cap T_e) \leq \text{mp}_{T_{e^*} \cap \text{OPT}}(e)$ on each edge $e \in E$. This can be done using a PTAS in [16] for *rooted* UFP instances in which all input tasks share a common edge (like the edge e^* in our case).

► **Theorem 6 ([16]).** *There is a PTAS for instances of UFP in which there is an edge that is used by every input task (rooted UFP). The same holds if there exist $O(1)$ edges such that each task uses at least one of them.*

We recurse on the subpaths on the left and on the right of e^* . Let us consider the left subpath, i.e., let E_L denote the path from the left-most edge of E up to e^* , not including e^* . (The recursion to the right is analogous.) We subdivide E_L into three parts determined by two edges e', e'' in E_L (or two parts if these edges coincide). We choose the first edge $e' \in E_L$ on the left of e^* such that the number of steps in the profile $\bar{u} := \text{mp}_{\text{OPT} \cap T_{e^*}}$ within E_L is halved (a similar trick was used in [7]). Formally, let $\Gamma = O_{\delta, \epsilon}(\log \log n)$ such that $\text{mp}_{\text{OPT} \cap T_{e^*}}$ is a step-function with Γ steps. We choose e' such that $\text{mp}_{\text{OPT} \cap T_{e^*}}$ restricted to the subpath of E_L on the left of e' is a step-function with at most $\lceil \Gamma/2 \rceil$ steps, and the same is true for $\text{mp}_{\text{OPT} \cap T_{e^*}}$ restricted to the subpath of E_L on the right of e' . The second edge e'' lies in the middle of E_L such that on the left of e'' there are at most $\lfloor |E_L|/2 \rfloor$ edges of E_L and the same is true on the right of e'' . We may assume without loss of generality that e' lies on the left of e'' .

We guess the profiles $\bar{u}' := \text{mp}_{(\text{OPT} \cap T_{e'}) \setminus T_{e^*}}$ and $\bar{u}'' := \text{mp}_{(\text{OPT} \cap T_{e''}) \setminus (T_{e^*} \cup T_{e'})}$ and apply Theorem 6 to compute essentially the most profitable subset of tasks that fit into \bar{u}' (and \bar{u}''), among the input tasks $i \in T_{e'}$ with $P(i) \subseteq E_L$ (among the input tasks $i \in T_{e''} \setminus T_{e'}$ with $P(i) \subseteq E_L$). Then we recurse on each of the up to three components of $E_L \setminus \{e', e''\}$, where the parameter is the respective subpath and the profile $\bar{u} + \bar{u}' + \bar{u}''$ restricted to that subpath. Due to the choice of e'' , the depth of our recursion is bounded by $D = \log n$. In particular, at the end each edge e is in the support of at most $O(D)$ guessed profiles. By Lemma 5, the total error of these profiles is at most $O(\delta)u(e)$ which is compensated by the resource augmentation. Due to the choice of e' , one can show that in each recursive call the profile of the parameter is a step function with only $4\Gamma = O_{\delta, \epsilon}(\log \log n)$ steps: each recursive call “inherits” only half of the steps that were given to its parent subproblem, and additionally up to 2Γ new steps due to the guessed profiles in the parent subproblem. This allows us to embed this recursion into a dynamic program (DP) with only $n^{O_{\delta, \epsilon}(\log \log n)}$ DP-cells as follows, and hence we obtain an overall running time of $n^{O_{\delta, \epsilon}(\log \log n)}$.

2.2 Dynamic program

We define the mentioned DP formally. In our DP table, we have a cell (P, \hat{u}) for each subpath $P \subseteq E$ and each function $\hat{u} : P \rightarrow \mathbb{N}_0$ such that $0 \leq \hat{u}(f) \leq n/(\delta\eta)$ for each edge $f \in P$ and \hat{u} is a step-function with at most 4Γ steps, i.e., one can partition P into 4Γ subpaths such that \hat{u} is constant on each one of them. The intuitive meaning of \hat{u} is that for each edge $e \in P$,

$\hat{u}(e)$ units of capacity have already been assigned to some tasks. The goal is to compute a set of tasks with maximum total weight and with $P(i) \subseteq P$ for each selected task i , such that on each edge $e \in P$ the selected tasks use a total capacity of at most $u(e) - \hat{u}(e)$.

► **Lemma 7.** *There are at most $n^{O_{\delta,\epsilon}(\log \log n)}$ many different DP cells.*

Proof. There are $O(n^2)$ possible choices for P . To determine the number of possible profiles \hat{u} , recall that \hat{u} has at most $4\Gamma = O_{\delta,\epsilon}(\log \log n)$ many steps. There are $\binom{O(n)}{O_{\delta,\epsilon}(\log \log n)} \in n^{O_{\delta,\epsilon}(\log \log n)}$ possibilities to partition P into at most $O_{\delta,\epsilon}(\log \log n)$ subpaths. For each subpath, there are at most $n/(\delta\eta)$ possibilities for the value of \hat{u} on this subpath, i.e., $(n/(\delta\eta))^{O_{\delta}(\log \log n)}$ possible combinations overall. Recall that $\eta = \delta^{2/\epsilon}$. Multiplying these two numbers gives the total number of profiles. ◀

Given a DP cell $\text{DP}(P, \hat{u})$. We identify two edges $e', e'' \in P$ similarly as above, i.e., we select $e' \in P$ such that if we restrict \hat{u} to the edges of P on the left of e' or on the right of e' , then \hat{u} is a step-function with at most 2Γ steps. Also, we select e'' such that at most $\lfloor |P|/2 \rfloor$ edges of P lie on the left of e'' , and at most $\lfloor |P|/2 \rfloor$ edges of P lie on the right of e'' . Let P_1, P_2, P_3 denote the subpaths of P induced by e' and e'' , i.e., the components of $P \setminus \{e', e''\}$.

Let \mathcal{T} be the set of all pairs (\hat{u}', \hat{u}'') such that $\hat{u}' : P \rightarrow \mathbb{N}_0$ and $\hat{u}'' : P \rightarrow \mathbb{N}_0$ are step-functions with at most Γ steps each, and on each edge $f \in P$ they use at most $(1 + \delta)u(f)$ units of capacity, i.e., $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) \leq (1 + \delta)u(f)$. We apply the algorithm due to Theorem 6 to the instance whose input tasks contain all tasks in $i \in T_{e'}$ with $P(i) \subseteq P$ and with edge capacities given by \hat{u}' . Similarly, we apply this algorithm to the instance whose input tasks contain all tasks in $i \in T_{e''} \setminus T_{e'}$ with $P(i) \subseteq P$ and with edge capacities given by \hat{u}'' . Denote by $\text{alg}(\hat{u}')$ and $\text{alg}(\hat{u}'')$ the respective solutions. With the pair (\hat{u}', \hat{u}'') we associate the solution given by $\text{alg}(\hat{u}') \cup \text{alg}(\hat{u}'')$ and the solutions stored in the cells $\text{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')|_{P_j})$ for $j \in \{1, 2, 3\}$. We store in $\text{DP}(P, \hat{u})$ the weight of the most profitable solution for all pairs $(\hat{u}', \hat{u}'') \in \mathcal{T}$, i.e.,

$$\text{DP}(P, \hat{u}) := \max_{t=(\hat{u}', \hat{u}'') \in \mathcal{T}} ((\text{alg}(\hat{u}')) + w(\text{alg}(\hat{u}'')) + \sum_{j=1}^3 \text{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')|_{P_j})).$$

Observe that $(\hat{u} + \hat{u}' + \hat{u}'')|_{P_j}$ has at most 4Γ steps as required for valid DP cells: the profile \hat{u} restricted to P_j has at most 2Γ steps due to the halving and both \hat{u}' and \hat{u}'' introduce at most Γ new steps each.

At the end we output $\text{DP}(E, u_0)$ where u_0 is the profile with $u_0(e) = 0$ for all $e \in E$. By standard memoization we can also output the solution associated with $\text{DP}(E, u_0)$.

We need to show that the DP computes a near-optimal solution. It is clear that we output a feasible solution since in the computation for each cell, the set \mathcal{T} contains only pairs (\hat{u}', \hat{u}'') such that $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) \leq (1 + \delta)u(f)$ for each edge $f \in P$. Intuitively, Lemma 5 allows us to argue that in each step we can guess for \hat{u}' and \hat{u}'' the profiles $\text{mp}_{\text{OPT}' \cap T_{e'}}$ and $\text{mp}_{(\text{OPT}' \cap T_{e''}) \setminus T_{e'}}$, respectively, where OPT' denotes the tasks $i \in \text{OPT}$ with $P(i) \subseteq P$, without violating the capacities of the edges.

► **Lemma 8.** *The dynamic program computes a solution ALG which is feasible with $O(\delta)$ -resource augmentation and $w(\text{ALG}) \geq (1 - \epsilon)w(\text{OPT})$.*

Proof. Consider a DP-cell (P, \hat{u}) for which the DP defined the edges e', e'' when calculating its solution. Consider the profiles $\text{mp}_{\text{OPT} \cap \{i \in T_{e'} \mid P(i) \subseteq P\}}$ and $\text{mp}_{\text{OPT} \cap \{i \in T_{e''} \setminus T_{e'} \mid P(i) \subseteq P\}}$. Observe that all tasks $\{i \in \text{OPT} \cap (T_{e'} \cup T_{e''}) \mid P(i) \subseteq P\}$ fit into the combined profile and all remaining tasks from OPT within P will be considered in subproblems. If the two profiles are a feasible choice, then the DP obtains a profit of at least $(1 - \epsilon)w(\{i \in \text{OPT} \cap (T_{e'} \cup T_{e''}) \mid P(i) \subseteq P\})$

from the tasks in $T_{e'} \cup T_{e''}$ when choosing these profiles. Therefore, it suffices to prove the claim that if in each recursive call up to some recursion level ℓ the DP chooses exactly these profiles, then in each subsequent recursive call of level $\ell + 1$ the corresponding profiles will be a feasible choice again. Inductively, we can conclude then that the DP obtains a profit of at least $(1 - \epsilon)\text{opt}$.

To prove the mentioned claim, intuitively, due to the halving of the paths, the recursion depth is bounded from above by $O(\log n)$ and therefore the total error does not exceed the available amount of resource augmentation. Formally, we inductively maintain the invariant that for a DP cell (P, \hat{u}) , if the profile \hat{u} is composed of $2k$ profiles (from previous DP cells), then the length of P is bounded from above by $2n/2^k$. Recall that we assumed that $|E| \leq O(n)$. For each edge, we therefore have that at most $O(\log n)$ profiles overlap and by Lemma 5, the total error of is at most $O(\delta)u(e)$ for each edge e , i.e., $O(\delta)$ -resource augmentation is sufficient. \blacktriangleleft

Together with Lemma 3 we conclude that we obtain a $(1 + \epsilon)$ -approximation algorithm for UFP with δ -resource augmentation.

3 Bonus-UFP

In this section we define formally the Bonus-UFP problem (BUFP) and formalize how an algorithm with resource augmentation for Bonus-UFP yields an algorithm for (ordinary) UFP *without* resource augmentation.

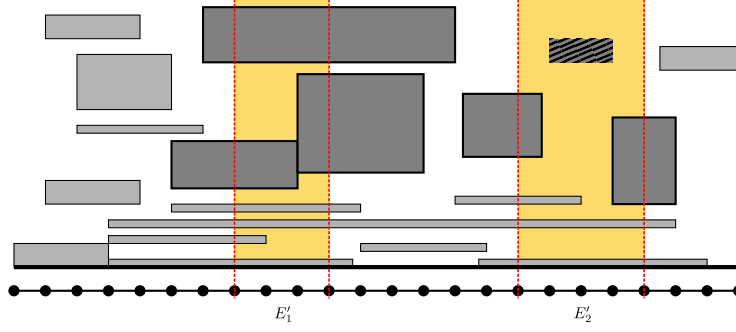
3.1 Problem definition

Like in (ordinary) UFP, in the Bonus-UFP problem we are given as input a path $G = (V, E)$ and a set of tasks T where each task $i \in T$ has a demand $d(i) \in \mathbb{N}$, a sub-path $P(i)$ of E , and a weight $w(i) \in \mathbb{N}_0$. The profit of a computed solution with task set $T' \subseteq T$ stems as usual from $w(T')$ and, additionally, from some *bonus profit* that we obtain from subpaths $E' \subseteq E$ (i.e., E' forms the edges of the respective subpath) such that no task $i \in T'$ satisfies that $P(i) \subseteq E'$ (but it might be that $P(i) \cap E' \neq \emptyset$). The amount of bonus of such a subpath E' depends on the tasks $i \in T'$ with $P(i) \cap E' \neq \emptyset$. The reader may imagine that we get more bonus from E' if fewer tasks from T' intersect E' . In particular, the computed solution consists of T' and of the subpaths E' from which the respective bonus profit is collected. Additionally, the input tasks are divided into large and small tasks, and on any edge we are allowed to select only $O(1)$ large tasks (in particular, for each edge e the large tasks in $OPT \cap T_e$ can be guessed easily in time $n^{O(1)}$). Also, for technical reasons, the selected small tasks are allowed to use at most $2b$ units of capacity of each edge $e \in E$ for some given value b (see Figure 2).

Formally, in the input we are given a constant $\tau \in \mathbb{N}$ and values $\mu \in (0, 1)$, $b > 0$ which partition the tasks T into a set of *large tasks* $T_L = \{i \in T : d(i) > \mu b\}$ and *small tasks* $T_S = \{i \in T : d(i) \leq \mu b\}$. For each pair of a subpath $E' \subseteq E$ and a set $L'_{int} \subseteq T_L$ such that

- the path of each task $i \in L'_{int}$ uses the leftmost or the rightmost edge of E' and
- $|L'_{int} \cap T_e| \leq \tau$ for each edge $e \in E'$

we are given a possible *bonus* $bn(E', L'_{int}) \geq 0$. As described above, we obtain the bonus $bn(E', L'_{int})$ if no selected task (large or small) is contained in E' and the tasks in L'_{int} are exactly the selected large tasks that use some edge of E' (the amount of bonus does not depend on the selected small tasks i with $P(i) \cap E' \neq \emptyset$). A feasible solution to an instance of BUFP consists of a set of tasks $R \subseteq T_L \cup T_S$ and a collection of node-disjoint subpaths E_1, \dots, E_q of G such that the following holds:



■ **Figure 2** A Bonus-UFP instance with two bonus intervals E' and E'' . The hatched task within E'' is not allowed. The bonus depends on the intervals and the tasks depicted in dark gray – the large tasks that intersect with an interval.

1. R is a feasible UFP solution, i.e., for every $e \in E$ it holds that $d(R \cap T_e) \leq u(e)$,
 2. the small tasks in R use at most a capacity of $2b$ on each edge $e \in E$, i.e., $d(R \cap T_S \cap T_e) \leq 2b$,
 3. each edge $e \in E$ is used by at most τ tasks in $R \cap T_L$, i.e., $|R \cap T_L \cap T_e| \leq \tau$, and
 4. for no $i \in R$ the path $P(i)$ is contained in any E_j (but possibly $P(i) \cap E_j \neq \emptyset$).
- The profit of the solution is $w(R)$ plus the bonus for each subpath E_j , where the latter depends on the large tasks in R that use E_j , i.e., the total profit is $w(R) + \sum_{j=1}^q bn(E_j, L_{int,j})$ where $L_{int,j} = \{i \in R \cap T_L \mid P(i) \cap E_j \neq \emptyset\}$.

We say that a solution to the above problem is feasible *under δ -resource augmentation* if we relax the first two conditions to

- 1.' for every $e \in E$ it holds that $d(R \cap T_e) \leq u(e) + \delta b$,
- 2.' for every $e \in E$ it holds that $d(R \cap T_S \cap T_e) \leq 2b + \delta b$.

So intuitively this increases the capacity for the small tasks by at least a factor $1 + O(\delta)$. Also, note that Property 3 ensures that each edge e is used by at most $\tau = O(1)$ large tasks, and hence we can guess the large tasks for each edge in time $n^{O(\tau)} = n^{O(1)}$.

For non-negative values $\alpha, \beta, \gamma \leq 1$ we say that a tri-criteria (α, β, γ) -*approximation algorithm for BUFP with δ -resource augmentation* is an algorithm that computes a solution that is feasible under δ -resource augmentation of profit at least $\alpha \text{opt}_S + \beta \text{opt}_L + \gamma \text{opt}_B$, assuming that there exists an (optimal) solution R^* with bonuses $\{bn(E_j^*, L_{int,j}^*)\}_j$ such that $\text{opt}_S = w(R^* \cap T_S)$, $\text{opt}_L = w(R^* \cap T_L)$, and $\text{opt}_B = \sum_{j=1}^q bn(E_j^*, L_{int,j}^*)$. Then our main result states that if $\gamma = 1$ we can translate such an algorithm to an algorithm for normal UFP whose approximation ratio is $1/\min\{\alpha, \beta\}$. We will prove the following theorem in Section 3.2.

► **Theorem 9** (Black-box reduction). *Assume that there is a $(\alpha, \beta, 1)$ -approximation algorithm for BUFP with δ -resource augmentation with a running time of $T_{\tau, \delta}(n, u_{\max})$. Then there is a $\frac{1+\epsilon}{\min\{\alpha, \beta\}}$ -approximation algorithm for UFP (without resource augmentation) with a running time of $T_{O_\epsilon(1), O_\epsilon(1)}(n, u_{\max}) \cdot (n \cdot \log u_{\max})^{O_\epsilon(1)}$.*

3.2 Black-box reduction

In this section we explain the key ideas for the black-box reduction due to Theorem 9 (omitting details due to space constraints). We start with a lemma indicating that there are near-optimal solutions in which each edge has a certain amount of slack. This slack is a

constant fraction of the capacity used by small tasks according to a suitable definition of small and large tasks. To avoid confusion, we refer to large and small tasks as in definition of BUFP as bonus-large and bonus-small, respectively.

In the lemma, we assign a level $\ell(e)$ to each edge e . We say that a task i is of level ℓ if $P(i)$ includes an edge e with $\ell(e) = \ell$ and no edge e' with $\ell(e') < \ell$; let $T^{(\ell)} \subseteq T$ denote all tasks of level ℓ . Intuitively, each edge e of level ℓ has an amount of slack $\varepsilon^4 a(\ell)$ for some value $a(\ell)$ corresponding to level ℓ . We define small and large tasks such that on each edge e of level ℓ , the small tasks of level ℓ use a total capacity of at most $2b(\ell)$ with $b(\ell) = O_\epsilon(a(\ell))$ and each edge e of level ℓ is used by at most $\tau = O_\epsilon(1)$ large tasks i with $d(i) = \Omega(a(\ell))$. Formally, for an offset $h' \in \{0, \dots, 1/\epsilon - 1\}$ and a value $h = \Theta(\frac{1}{\epsilon} \ln \frac{7}{\epsilon^2})$ defined in the next lemma, we set $a(\ell) := (1 + \varepsilon)^{h' + \ell h \cdot (1 + \frac{1}{\epsilon}) - \frac{h}{\epsilon}}$ and $b(\ell) = a(\ell) \cdot (1 + \varepsilon)^{\frac{h}{\epsilon}}$. Let opt denote the weight of the optimal solution to the given instance.

► **Lemma 10 (Slack Lemma).** *Let $\epsilon > 0$. There are two constants $\mu_1, \mu_2 \in (0, \varepsilon^4)$ with $\mu_1 < \mu_2 / (1 + \varepsilon)^{1/\varepsilon^3}$, values $h = \Theta(\frac{1}{\epsilon} \ln \frac{7}{\epsilon^2})$ and $\tau = O_\epsilon(1)$, an offset h' contained in a set of size $O_\epsilon(1)$ that can be computed in polynomial time, a near-optimal solution OPT with $w(\text{OPT}) \geq (1 - O(\epsilon))\text{opt}$, and a level $\ell(e)$ for each edge $e \in E$ with the following properties. We define*

- $T_L := \{i \in T : d(i) \geq \mu_2 \cdot a(\ell(e)) \text{ for some edge } e \in P(i)\}$ (large tasks)
- $T_S := \{i \in T : d(i) < \mu_1 \cdot b(\ell(e)) \text{ for every edge } e \in P(i)\}$ (small tasks)
- $\text{OPT}_L = \text{OPT} \cap T_L$ and $\text{OPT}_S^{(\ell)} := \text{OPT} \cap T_S \cap T^{(\ell)}$ for each ℓ .

Then, for each edge e of level $\ell(e) = \ell$, it holds that:

- $d(T_e \cap \text{OPT}_L) + d(T_e \cap \text{OPT}_S^{(\ell)}) \leq u(e) - \varepsilon^4 a(\ell)$,
- $d(T_e \cap \text{OPT}_S^{(\ell)}) \leq 2b(\ell)$,
- e is used by at most τ tasks $i \in \text{OPT}_L \cap T^{(\ell)}$ with $d(i) \geq \mu_2 \cdot a(\ell)$.

Our goal is to define an algorithm BB that solves UFP given an oracle OR for BUFP with δ -resource augmentation. We will use the notation $G^B, T^B, T_S^B, T_L^B, u^B, \tau^B$ for the input path, input tasks, edge capacities, and threshold τ of the constructed instances of BUFP, while we use G, T, u to denote the corresponding values of the given instance of UFP.

Intuitively, BB solves a subproblem of BUFP for each maximally long path G^B in which each edge is of level at least ℓ for some given value $\ell \in \{0, \dots, \ell_{\max}\}$ (where ℓ_{\max} denotes the maximum level due to Lemma 10), and the bonus subpaths E_j selected in an optimal solution will correspond to maximal subpaths of G^B consisting of edges of level at least $\ell + 1$. With this interpretation, the amount of received bonus on such a subpath E_j is calculated via subproblems of BUFP defined on this subpath E_j , level $\ell + 1$, and any possible set of $O_\epsilon(1)$ selected large tasks of level ℓ that use at least one edge of E_j but which are not contained in E_j . The parameters of the calls to OR for this a subproblem (i.e., path G^B and level ℓ) will be intuitively as follows. We set $b = b(\ell)$, $\mu = \mu_1$, and $a = a(\ell)$. We place in T^B the tasks in T contained in G^B whose demand is at most $\mu_1 b(\ell)$ or at least $\mu_2 a(\ell)$. This yields that $T_L^B = \{i \in T^B | d(i) \geq \mu_2 a(\ell)\}$ and $T_S^B = \{i \in T^B | d(i) \leq \mu_1 b(\ell)\}$ according to the definition of BUFP. We remark that either $\mu_2 a(\ell) > \mu_1 b(\ell)$ or $\mu_1 b(\ell) < 1$ (in which case $T_L^B = \emptyset$); hence T_L^B and T_S^B are distinct. This way we will enforce that $T_L^B \subseteq T_L$ and $T_S^B \subseteq T_S$ for the sets T_L, T_S due to Lemma 10. We will set τ^B to be the respective value τ due to Lemma 10.

Note that in this recursive call there are $O_\epsilon(1)$ (previously selected) large tasks that use some but possibly not all edges of E_j . However, in the definition of Bonus-UFP, we have a global upper bound of τ^B for the number of allowed large tasks using an edge. To this end, we guess some further large tasks whose paths are contained in E_j , profiles for some of the

small tasks, and a partition of E_j into $O_\epsilon(1)$ subpaths, such that we split this problem into subproblems in which each edge can be used by the same number $\tau^B \leq \tau$ of large tasks, so that we can call OR on the resulting subproblem. We denote by opt_S and opt_L the profit due to small and large tasks, resp., in the solution OPT from Lemma 10.

► **Theorem 11.** *Given a $T_{\tau,\delta}(n, u_{\max})$ time $(\alpha, \beta, 1)$ -approximation algorithm OR for BUFP with δ -resource augmentation, for every given constant $\delta > 0$. Then, for every constant $\epsilon > 0$, there exists a $T_{O_\epsilon(1), O_\epsilon(1)}(n, u_{\max}) \cdot (n \log u_{\max})^{O_\epsilon(1)}$ time algorithm BB for UFP that computes a solution of profit at least $(1 - \epsilon)\alpha \text{opt}_S + (1 - \epsilon)\beta \text{opt}_L$.*

Theorem 9 then follows from Lemma 10 and Theorem 11, where we set $\alpha = 1 - \epsilon$ and $\beta = 1$.

The reader might wonder why we proved the above theorem for general α and β . The reason is that it turns out that tasks $i \in OPT_S = OPT \cap T_S$ from Lemma 10 satisfy $d(i) \leq \epsilon^2 u(e)$ for each $e \in P(i)$. Therefore, we can use LP-rounding techniques as in [11] to compute an alternative UFP solution with profit at least $(1 - O(\epsilon))\text{opt}_S$. Hence any algorithm for BUFP as in Theorem 11 implies a $(\frac{1+\beta-\alpha}{\beta} + O(\epsilon))$ -approximation for UFP. Recent work showed how to obtain (α, β) -approximation algorithms for UFP in the above sense with $\beta = 1$ and α equal to $\frac{1}{3}$ [17]. This result becomes substantially easier to prove with resource augmentation in the BUFP setting. So we hope that Theorem 11 as stated can be a handy tool for future work along the same line.

4 Algorithm for Bonus-UFP under resource augmentation

We describe now how to adjust our algorithm for ordinary UFP with resource augmentation from Section 2 to an algorithm for Bonus-UFP with resource augmentation. Intuitively, there are two extra issues than one needs to address. First, the edges e' and e'' that partition the considered subpath P might happen to fall within the interval of some bonus. This has to be taken into account in the definition of the subproblems which are solved recursively. Second, we need to keep track of the large tasks that use each edge so that the threshold τ is not exceeded. Therefore it is convenient to *guess* them explicitly, and use the over-estimated profiles for the small tasks only.

► **Theorem 12.** *There is a $(1 - \epsilon, 1, 1)$ -approximation algorithm for Bonus-UFP with δ -resource augmentation with a running time of $n^{O_{\epsilon,\delta}(\log \log n)}$.*

Theorems 9 and 12 together yield Theorem 2. It remains a challenging open problem to construct a PTAS for (Bonus-)UFP. One key bottleneck in our approach is that already in the first iteration we need to guess a profile with up to $\Omega(\log \log n)$ many steps, and it is not clear how to do this in polynomial time.

In the remaining section, we prove Theorem 12. Again, we start with normalizing the instance. Since we have an additive resource augmentation of δb , we can use half of it in order to ensure $u_{\min} \geq \delta b/2$. Thus $2b \leq 4u_{\min}/\delta$, i.e., $u_{OPT_S}(e)$ together with half of the resource augmentation has a constant capacity profile. Furthermore, analogous to Lemma 3, we can assume that each task $i \in T$ has an integer demand $d(i) \in \mathbb{N}$ and $b \in O_\delta(n)$. Unlike before, we do not lose a profit of $\epsilon w(\text{OPT}_L)$.

► **Lemma 13.** *For arbitrary $\delta > 0$, suppose there is an (α, β, γ) -approximation algorithm for BUFP instances I with $(1 + \delta)$ resource augmentation such that $u_{\min} = \delta b$, $b \in O_\delta(n)$, and $d(i) \in \mathbb{N}$ for all $i \in T$. Then there is an $((1 - \epsilon)\alpha, \beta, \gamma)$ -approximation algorithm for BUFP with $(1 + 4\delta)$ resource augmentation.*

49:12 Faster $(1 + \epsilon)$ -Approximation for Unsplittable Flow on a Path

Due to Lemma 13, in the following we assume that we are given an instance of BUFP with $(1 + \delta)$ resource augmentation such that $u_{\min} = \delta b$, $2b = n/\eta$ for some $\eta \in O_\delta(1)$, and $d(i) \in \mathbb{N}$ for all $i \in T$.

We adapt the strategy used for UFP with resource augmentation and we use the same notation. Again we overestimate profiles based on the value $\theta := \lfloor u_{\min} \cdot \delta / D \rfloor$, now with $u_{\min} = \delta b$ and $D = \log n$. Lemma 4 is still valid for BUFP, if we restrict the set S to small tasks.

► **Lemma 14.** *There is a value $\Gamma = O_{\delta,\epsilon}(\log D) = O_{\delta,\epsilon}(\log \log n)$ such that for every edge $\bar{e} \in E$ and every set $S \subseteq T_S \cap T_{\bar{e}}$ with $d(S \cap T_e) \leq 2b$ for each $e \in E$, we have that $mp_S(\cdot)$ is a step-function with at most Γ steps.*

Proof. Since the demand $d(S)$ is bounded from above by $2b$, it is sufficient to consider the profile that for each edge $e \in E$ has the capacity $\min\{2b, u(e)\}$. With the modification, we can apply Lemma 4. ◀

Furthermore, Lemma 5 does not change for BUFP, if we restrict it to small tasks, i.e., $S_j \subseteq T_S$ for all j .

For ease of notation, we assume that for each edge f and each set $L \subseteq T_L \cap T_e$ with $d(L) \leq u(f)$, the BUFP instance to be solved has is a (dummy) bonus interval $(\{f\}, L)$. If the bonus interval is not part of the original instance, it has zero bonus, i.e., $bn(\{f\}, L) = 0$. By duplicating edges, we can assume without loss of generality that there is no task i with $P(i) = f$.

4.1 Dynamic program for BUFP

In our DP table, we have a cell (P, \hat{u}, L) for each subpath $P \subseteq E$, each function $\hat{u} : P \rightarrow \mathbb{N}_0$, and each set of large tasks $L \subseteq T_L$ with the following properties. We require $0 \leq \hat{u}(f) \leq n/\eta$ for each edge $f \in P$ and \hat{u} is a step-function with at most 8Γ steps, i.e., one can partition P into 8Γ subpaths such that \hat{u} is constant on each of them. A set L is valid, if each task in L uses the leftmost or the rightmost edge of P . Note that $|L| \leq 2\tau$ since there can only be at most τ large tasks crossing each of the two boundaries of P .

The intuitive meaning of \hat{u} and L is that for each edge $e \in P$, $\hat{u}(e)$ units of capacity have already been assigned to some small tasks and the tasks L have already been selected. For a set of tasks T' , we define $u_{T'} : E \rightarrow \mathbb{N}_0$ to be the profile with $u_{T'}(e) = d(T' \cap T_e)$ for each $e \in E$. The goal is to compute a set of tasks with maximum total weight and with $P(i) \subseteq P$ for each selected task i , such that on each edge $e \in P$ the selected tasks use a total capacity of at most $u(e) - \hat{u}(e) - u_L(e)$.

► **Lemma 15.** *There are at most $n^{O_{\delta,\epsilon,\tau}(\log \log n)}$ many different DP cells.*

Proof. There are $O(n^2)$ possible choices for P . To determine the number of possible profiles \hat{u} , recall that \hat{u} has at most $8\Gamma = O_{\delta,\epsilon}(\log \log n)$ many steps and L has a size of at most 2τ .

There are $\binom{O(n)}{O_{\delta,\epsilon,\tau}(\log \log n)} \in n^{O_{\delta,\epsilon,\tau}(\log \log n)}$ possibilities to partition P into at most $O_{\delta,\epsilon,\tau}(\log \log n)$ subpaths. For each subpath, there are at most n/η possibilities for the value of \hat{u} on this subpath and $\binom{n}{2\tau} \leq n^{2\tau}$ possibilities for selecting L , i.e., $(n/\eta)^{O_{\delta,\epsilon}(\log \log n)} \cdot n^{2\tau}$ possible combinations overall. Multiplying these two numbers gives the total number of profiles. ◀

Suppose that we are given a DP cell $DP(P, \hat{u}, L)$. We identify two edges $e', e'' \in P$. We select $e' \in P$ such that if we restrict \hat{u} to the edges of P on the left of e' or on the right of e' , then \hat{u} is a step-function with at most 4Γ steps. Also, we select e'' such that at most

$\lfloor |P|/2 \rfloor$ edges of P lie on the left of e'' , and at most $\lfloor |P|/2 \rfloor$ edges of P lie on the right of e'' . Let P_1, P_2, P_3 denote the subpaths of P induced by $E_{j'}$ and $E_{j''}$, i.e., the components of $P \setminus (E_{j'} \cup E_{j''})$.

Let \mathcal{T} be the set of all tuples $(p_1, p_2, \hat{u}', \hat{u}'')$ with $p_1 := (E_{j'}, L_{\text{int},j'})$ and $p_2 := (E_{j''}, L_{\text{int},j''})$ specified as follows. The pair $(E_{j'}, L_{\text{int},j'})$ is a bonus interval with $e' \in E_{j'}$ and $(E_{j''}, L_{\text{int},j''})$ is a bonus interval with $e'' \in E_{j''}$ such that $E_{j'} \cap E_{j''} = \emptyset$ or $p_1 = p_2$. A task i which overlaps with both $E_{j'}$ and $E_{j''}$ (i.e., $P(i) \cap E_{j'} \neq \emptyset$ and $P(i) \cap E_{j''} \neq \emptyset$) is either in both $L_{\text{int},j'}$ and $L_{\text{int},j''}$ or in none.

Each of the remaining two entries of the tuple $\hat{u}' : P \rightarrow \mathbb{N}_0$ and $\hat{u}'' : P \rightarrow \mathbb{N}_0$ is composed of two step-functions with at most Γ steps each, and on each edge $f \in P$, $\hat{u}'(f) + \hat{u}''(f) \leq (2 + \delta)b$. Furthermore, the capacity of all step-functions use at most $u(f) + \delta b$ units of capacity, i.e., $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) + u_L(f) + u_{L_{\text{int},j'}}(f) + u_{L_{\text{int},j''}}(f) \leq u(f) + \delta b$. Let $e'_\ell, e'_r, e''_\ell, e''_r$ be the left-most and right-most edge of $E_{j'}$ and $E_{j''}$, respectively. We apply the algorithm due to Theorem 6 to the instance whose input tasks contain all tasks in $i \in T_{e'_\ell} \cup T_{e'_r}$ with $P(i) \subseteq P$ and $P(i) \not\subseteq E_{j'}$, with edge capacities given by \hat{u}' . Similarly, we apply this algorithm to the instance whose input tasks contain all tasks in $i \in T_{e''_\ell} \cup T_{e''_r} \setminus (T_{e'_\ell} \cup T_{e'_r})$ with $P(i) \subseteq P$ and $P(i) \not\subseteq E_{j''}$, with edge capacities given by \hat{u}'' . Denote by $\text{alg}(\hat{u}')$ and $\text{alg}(\hat{u}'')$ the respective solutions. With the tuple $(p_1, p_2, \hat{u}', \hat{u}'')$ we associate the solution given by $\text{alg}(\hat{u}') \cup \text{alg}(\hat{u}'')$, the bonuses $\text{bn}(p_1), \text{bn}(p_2)$, the profit from large tasks $w(L_{\text{int},j'} \cup L_{\text{int},j''} \setminus L)$, and the solutions stored in the cells $\text{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')|_{P_j}, L')$ for $j \in \{1, 2, 3\}$ and L' the tasks from $L \cup L_{\text{int},j'} \cup L_{\text{int},j''}$ crossing P_j . We store in $\text{DP}(P, \hat{u}, L)$ the weight of the most profitable solution for all tuples $(p_1, p_2, \hat{u}', \hat{u}'') \in \mathcal{T}$, i.e.,

$$\begin{aligned} \text{DP}(P, \hat{u}, L) := & \max_{t=(p_1, p_2, \hat{u}'_S, \hat{u}''_S) \in \mathcal{T}} (w(\text{alg}(\hat{u}')) + w(\text{alg}(\hat{u}''))) \\ & + \sum_{p \in \{p_1, p_2\}} \text{bn}(p) + w(L_{\text{int},j'} \cup L_{\text{int},j''} \setminus L) + \sum_{j=1}^3 \text{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')|_{P_j}), \end{aligned}$$

where \hat{u}' and \hat{u}'' are derived from t as described before. Observe that if $p_1 = p_2$, we collect only one bonus. We have to ensure that $\hat{u}^{(j)} := (\hat{u} + \hat{u}' + \hat{u}'')|_{P_j}$ is a valid profile. Due to the halving, restricted to P_j the profile \hat{u} has at most 4τ steps and each of the other two profiles has at most 2Γ steps from small tasks, which results in 8Γ steps in total. Independently, we always have a total number of at most 4τ steps from large tasks

At the end we output $\text{DP}(E, u_0)$ where u_0 is the profile with $u_0(e) = 0$ for all $e \in E$. By standard memoization we can also output the solution associated with $\text{DP}(E, u_0)$.

We need to show that the DP computes a near-optimal solution. It is clear that we always output a feasible solution since in the computation for each cell, the set \mathcal{T} contains only tuples $(\hat{u}'_S, \hat{u}''_S, p_1, p_2)$ such that $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) + u_L(f) + u_{L_{\text{int},j'}}(f) + u_{L_{\text{int},j''}}(f) \leq u(f) + \delta b$ for each edge $f \in P$. Intuitively, Lemma 5 allows us to argue that in each step we can guess for \hat{u}' and \hat{u}'' , the pairs p_1 and p_2 , the profiles $\text{mp}_{\text{OPT}' \cap T_{e'}}$ and $\text{mp}_{(\text{OPT}' \cap T_{e''}) \setminus T_{e'}}$, respectively, where OPT' denotes the tasks $i \in \text{OPT}$ with $P(i) \subseteq P$, without violating the capacities of the edges. Let OPT_B be the set of bonus pairs of an optimal solution and $w(\text{OPT}_B)$ the sum of bonuses.

► **Lemma 16.** *The dynamic program computes a solution ALG which is feasible with $O(\delta)$ -resource augmentation and $w(\text{ALG}) \geq (1 - \epsilon)w(\text{OPT}_S) + w(\text{OPT}_L) + w(\text{OPT}_B)$.*

Proof. Initially, the DP chooses the middle edge e (i.e., $e = e' = e''$) and a bonus interval containing e . One of the choices is the bonus pair $p = (E_j, L_{\text{int},j})$ such that $p \in \text{OPT}_B$ and $e \in E_j$. Since OPT is feasible, it has no tasks contained in E_j . Let e_ℓ and e_r be the left-most and right-most edge of E_j . Then one of the options for the profile is to choose

$$\hat{u} := \text{mp}_{\text{OPT} \cap T_{e_\ell}} + \text{mp}_{\text{OPT} \cap T_{e_r}}.$$

All tasks $\text{OPT} \cap (T_{e_\ell} \cup T_{e_r})$ fit into the profile \hat{u} . Since the DP approximates the profit from these tasks, the value of the DP cell without the values from the subproblems is at least $(1 - \epsilon)w(\text{OPT}_S \cap T_e) + w(\text{OPT}_L \cap (T_{e_\ell} \cup T_{e_r})) + \text{bn}(p)$. The instance is split into the left hand side of E_j and the right hand side of E_j . Let (P, \hat{u}, L) be a sub-problem reached (recursively) by the described choice of profile. Each edge e' chosen subsequently within P has the property that the DP can choose the bonus pair $p_1 := (E_{j'}, L_{\text{int},j'})$ (where as before we assume that e'_ℓ and e'_r are the leftmost and rightmost edges of $E_{j'}$, respectively) from OPT_B containing e' and a feasible choice of \hat{u}' is $\text{mp}_{\text{OPT}_S \cap \{i \in T_{e'_\ell} \cup T_{e'_r} \mid P(i) \subseteq P\}}$. Each edge e'' chosen subsequently within P has the property that the DP can choose the bonus pair $p_2 := (E_{j''}, L_{\text{int},j''})$ (where as before we assume that e''_ℓ and e''_r are the leftmost and rightmost edges of $E_{j''}$, respectively) from OPT_B containing e'' and a feasible choice of \hat{u}' is $\text{mp}_{\text{OPT}_S \cap \{i \in T_{e'_\ell} \cup T_{e'_r} \setminus (T_{e''_\ell} \cup T_{e''_r}) \mid P(i) \subseteq P\}}$. Observe that all tasks $\{i \in \text{OPT}_S \cap (T_{e'_\ell} \cup T_{e'_r} \cup T_{e''_\ell} \cup T_{e''_r}) \mid P(i) \subseteq P\}$ fit into the combined profiles, we select all tasks specified in p_1 and p_2 , and all remaining tasks from OPT within P will be available in sub-problems. The DP collects a profit of at least $(1 - \epsilon)w(\{i \in \text{OPT}_S \cap (T_{e'_\ell} \cup T_{e'_r} \cup T_{e''_\ell} \cup T_{e''_r}) \mid P(i) \subseteq P\}) + w(\{i \in \text{OPT}_L \cap (T_{e'_\ell} \cup T_{e'_r} \cup T_{e''_\ell} \cup T_{e''_r}) \setminus L \mid P(i) \subseteq P\}) + \text{bn}(p_1) + \text{bn}(p_2)$. We inductively conclude that if all choices as described above are feasible, the DP obtains a profit of at least $(1 - \epsilon)w(\text{OPT}_S) + w(\text{OPT}_L) + w(\text{OPT}_B)$.

To show feasibility, we have to argue that none of the described choices exceeds the overall capacity of $u(f) + \delta b$ for an edge $f \in E$. The argument is analogous to the proof of Lemma 8. \blacktriangleleft

Together with Lemma 13 we conclude that we obtain a $(1 - \epsilon, 1, 1)$ -approximation algorithm for BUFP with δ -resource augmentation.

References

- 1 Anna Adamaszek, Parinya Chalermsook, Alina Ene, and Andreas Wiese. Submodular unsplittable flow on trees. *Math. Program.*, 172(1-2):565–589, 2018. doi:10.1007/s10107-017-1218-4.
- 2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013. doi:10.1007/978-3-642-36694-9_3.
- 3 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing $2 + \epsilon$ approximation for unsplittable flow on a path. *ACM Trans. Algorithms*, 14(4):55:1–55:23, 2018. doi:10.1145/3242769.
- 4 Y. Azar and O. Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006. doi:10.1007/s00453-005-1172-z.
- 5 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.
- 6 N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.
- 7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

- 8 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.
- 9 A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007.
- 10 C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.
- 11 C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.
- 12 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.
- 13 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 820–829. IEEE, 2016.
- 14 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, pages 13–24, 2015.
- 15 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Unsplittable flow on a path: The game!, 2020. unpublished.
- 16 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 *SODA*, pages 2411–2422, 2017.
- 17 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*, pages 607–619, 2018. doi:10.1145/3188745.3188894.
- 18 Shi Li. Towards PTAS for precedence constrained scheduling via combinatorial algorithms. In *SODA*, pages 2991–3010. SIAM, 2021.
- 19 Andreas Wiese. A $(1+\epsilon)$ -approximation for unsplittable flow on a path in fixed-parameter running time. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 67:1–67:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.

Quantum Sub-Gaussian Mean Estimator

Yassine Hamoudi 

Université de Paris, IRIF, CNRS, F-75013 Paris, France

Abstract

We present a new quantum algorithm for estimating the mean of a real-valued random variable obtained as the output of a quantum computation. Our estimator achieves a nearly-optimal quadratic speedup over the number of classical i.i.d. samples needed to estimate the mean of a heavy-tailed distribution with a sub-Gaussian error rate. This result subsumes (up to logarithmic factors) earlier works on the mean estimation problem that were not optimal for heavy-tailed distributions [9, 8], or that require prior information on the variance [23, 32, 22]. As an application, we obtain new quantum algorithms for the (ϵ, δ) -approximation problem with an optimal dependence on the coefficient of variation of the input random variable.

2012 ACM Subject Classification Theory of computation \rightarrow Quantum computation theory

Keywords and phrases Quantum algorithm, statistical analysis, mean estimator, sub-Gaussian estimator, (ϵ, δ) -approximation, lower bound

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.50

Funding This research was supported in part by the ERA-NET Cofund in Quantum Technologies project QuantAlgo and the French ANR Blanc project RDAM.

Acknowledgements The authors want to thank the anonymous referees for their valuable comments and suggestions which helped to improve this paper.

1 Introduction

The problem of estimating the mean μ of a real-valued random variable X given *i.i.d.* samples from it is one of the most basic tasks in statistics and in the Monte Carlo method. The properties of the various classical *mean estimators* are well understood. The standard non-asymptotic criterion used to assess the quality of an estimator is formulated as the following *high probability deviation bound*: upon performing n random experiments that return n samples from X , and given a failure probability $\delta \in (0, 1)$, what is the smallest error $\epsilon(n, \delta, X)$ such that the output $\tilde{\mu}$ of the estimator satisfies $|\tilde{\mu} - \mu| > \epsilon(n, \delta, X)$ with probability at most δ ? Under the standard assumption that the unknown random variable X has a finite variance σ^2 , the best possible performances are obtained by the so-called *sub-Gaussian estimators* [30] that achieve the following deviation bound

$$\Pr \left[|\tilde{\mu} - \mu| > L \sqrt{\frac{\sigma^2 \log(1/\delta)}{n}} \right] \leq \delta \quad (1)$$

for some constant L . The term “sub-Gaussian” reflects that these estimators have a Gaussian tail even for non-Gaussian distributions. The most well-known sub-Gaussian estimator is arguably the *median-of-means* [35, 27, 2], which consists of partitioning the n samples into roughly $\log(1/\delta)$ groups of equal size, computing the empirical mean over each group, and returning the median of the obtained means.

The process of generating a random sample from X is generalized in the quantum model by assuming the existence of a unitary operator U where $U|0\rangle$ coherently encodes the distribution of X . A *quantum experiment* is then defined as one application of this operator or its inverse. The celebrated quantum amplitude estimation algorithm [9] provides a way to estimate the mean of any *Bernoulli* random variable by performing fewer experiments



© Yassine Hamoudi;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 50; pp. 50:1–50:17

Leibniz International Proceedings in Informatics



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

than with any classical estimator. Yet, for general distributions, the existing quantum mean estimators either require additional information on the variance [23, 32, 22] or are less performant than the classical sub-Gaussian estimators when the distribution is heavy tailed [9, 38, 8, 32]. These results leave open the existence of a general quantum speedup for the mean estimation problem. We address this question by introducing the concept of *quantum sub-Gaussian estimators*, defined through the following deviation bound

$$\Pr\left[|\tilde{\mu} - \mu| > L \frac{\sigma \log(1/\delta)}{n}\right] \leq \delta \quad (2)$$

for some constant L . We give the first construction of a quantum estimator that achieves this bound up to a logarithmic factor in n . Additionally, we prove that it is impossible to go below that deviation level. This result provides a clear equivalent of the concept of sub-Gaussian estimator in the quantum setting.

A second important family of mean estimators addresses the (ϵ, δ) -approximation problem, where given a fixed relative error $\epsilon \in (0, 1)$ and a failure probability $\delta \in (0, 1)$ the goal is to output a mean estimate $\tilde{\mu}$ such that

$$\Pr[|\tilde{\mu} - \mu| > \epsilon|\mu|] \leq \delta. \quad (3)$$

The aforementioned sub-Gaussian estimators do not quite answer this question since the number of experiments they require (respectively $n = \Omega((\frac{\sigma}{\epsilon\mu})^2 \log(1/\delta))$ and $n = \tilde{\Omega}(\frac{\sigma}{\epsilon|\mu|} \log(1/\delta))$) depends on the *unknown* quantities σ and μ . Sometimes a good upper bound is known on the *coefficient of variation* $|\sigma/\mu|$ and can be used to parametrize a sub-Gaussian estimator. Otherwise, the standard approach is based on *sequential analysis* techniques, where the number of experiments is chosen adaptively depending on the results of previous computations. Given a random variable distributed in $[0, 1]$, the optimal classical estimators perform $\Theta((\frac{\sigma}{\epsilon\mu})^2 + \frac{1}{\epsilon\mu}) \log(1/\delta)$ random experiments *in expectation* [17] for computing an (ϵ, δ) -approximation of μ . We construct a quantum estimator that reduces this number to $\tilde{\Theta}((\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}) \log(1/\delta))$ and we prove that it is optimal.

1.1 Related work

There is an extensive literature on classical sub-Gaussian estimators and we refer the reader to [30, 15, 12, 18, 28] for an overview of the main results and recent improvements. We point out that the *empirical mean* estimator is not sub-Gaussian, although it is optimal for Gaussian random variables [37, 15]. The non-asymptotic performances of the empirical mean estimator are captured by several standard concentration bounds such as the Chebyshev, Chernoff and Bernstein inequalities.

There is a series of quantum mean estimators [21, 1, 8] that get close to the bound $\Pr\left[|\tilde{\mu} - \mu| > L \frac{\log(1/\delta)}{n}\right] \leq \delta$ for any random variable distributed in $[0, 1]$ and some constant L . Similar results hold for numerical integration problems [1, 36, 23, 39, 24]. The amplitude estimation algorithm [9, 38] leads to a sharper bound of $\Pr\left[|\tilde{\mu} - \mu| > L\left(\frac{\sqrt{\mu(1-\mu)} \log(1/\delta)}{n} + \frac{\log(1/\delta)^2}{n^2}\right)\right] \leq \delta$ (see Proposition 12) when X is distributed in $[0, 1]$. Nevertheless, the quantity $\mu(1-\mu)$ is always larger than or equal to the variance σ^2 . The question of improving the dependence on σ^2 was considered in [23, 32, 22]. The estimators of [23, 32] require to know an upper bound Σ on the standard deviation σ , whereas [22] needs an upper bound Δ on the coefficient of variation σ/μ (for non-negative random variables). The performances of these estimators are captured (up to logarithmic factors) by the deviation bound given in Equation (2) with σ replaced by Σ and $\mu\Delta$ respectively.

The (ϵ, δ) -approximation problem has been addressed by several classical works such as [17, 31, 20, 26]. In the quantum setting, there is a variant [9, Theorem 15] of the amplitude estimation algorithm that performs $O(\log(1/\delta)/(\epsilon\sqrt{\mu}))$ experiments in expectation to compute an (ϵ, δ) -approximate of the mean of a random variable distributed in $[0, 1]$ (see Theorem 7 and Proposition 15). However, the complexity of this estimator does not scale with σ . Given an upper bound Δ on σ/μ , the estimator of [22] can be used to compute an (ϵ, δ) -approximate with roughly $\tilde{O}(\Delta \log(1/\delta)/\epsilon)$ quantum experiments if the random variable is non-negative.

We note that the related problem of estimating the mean with *additive* error ϵ , that is $\Pr[|\tilde{\mu} - \mu| > \epsilon] \leq \delta$, has also been considered by several authors. The optimal number of experiments is $\Theta(\log(1/\delta)/\epsilon^2)$ classically [14] and $\Theta(1/\epsilon)$ quantumly [34] (with failure probability $\delta = 1/3$). These bounds do not depend on unknown parameters (as opposed to the relative error case), thus sequential analysis techniques are unnecessary here. Montanaro [32] also described an estimator that performs $\tilde{O}(\Sigma \log(1/\delta)/\epsilon)$ quantum experiments given an upper bound Σ on the standard deviation σ .

1.2 Contributions and organization

We first formally define the input model in Section 2.1. We introduce the concept of “q-random variable” (Definition 3) to describe a random variable that corresponds to the output of a quantum computation. We measure the complexity of an algorithm by counting the number of *quantum experiments* (Definition 4) it performs with respect to a q-random variable. We also introduce some needed tools in Section 2.2. Next, we construct a quantum algorithm for estimating the quantiles of a q-random variable in Section 3, and we use it in Section 4 to design the following quantum sub-Gaussian estimator.

Theorem 13 (Restated). *There exists a quantum algorithm with the following properties. Let X be a q-random variable with mean μ and variance σ^2 , and set as input a time parameter n and a real $\delta \in (0, 1)$ such that $n \geq \log(1/\delta)$. Then, the algorithm outputs a mean estimate $\tilde{\mu}$ such that $\Pr\left[|\tilde{\mu} - \mu| > \frac{\sigma \log(1/\delta)}{n}\right] \leq \delta$ and it performs $O(n \log^{3/2}(n) \log \log(n))$ quantum experiments.*

Then we turn our attention to the (ϵ, δ) -approximation problem in Section 5. In case we have an upper bound Δ on the coefficient of variation $|\sigma/\mu|$, we directly use our sub-Gaussian estimator to obtain an algorithm that performs $\tilde{O}(\frac{\Delta}{\epsilon} \log(1/\delta))$ quantum experiments (Corollary 14). Next, we consider the more subtle parameter-free setting where there is no prior information about the input random variable, except that it is distributed in $[0, 1]$. In this case, the number of experiments is chosen *adaptively*, and the bound we get is stated in expectation.

Theorem 16 (Restated). *There exists a quantum algorithm with the following properties. Let X be a q-random variable distributed in $[0, 1]$ with mean μ and variance σ^2 , and set as input two reals $\epsilon, \delta \in (0, 1)$. Then, the algorithm outputs a mean estimate $\tilde{\mu}$ such that $\Pr[|\tilde{\mu} - \mu| > \epsilon\mu] \leq \delta$, and it performs $\tilde{O}\left(\left(\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}\right) \log(1/\delta)\right)$ quantum experiments in expectation.*

Finally, we prove several lower bounds in Section 6 that match the complexity of the above estimators. We also consider the weaker input model where one is given copies of a quantum state encoding the distribution of X . We prove that no quantum speedup is achievable in this setting (Theorem 22).

1.3 Proof overview

Sub-Gaussian estimator. Our approach (Theorem 13) combines several ideas used in previous classical and quantum mean estimators. In this section, we simplify the exposition by assuming that the random variable X is non-negative and by replacing the variance σ^2 with the second moment $\mathbb{E}[X^2]$. We also take the failure probability δ to be a small constant. Our starting point is a variant of the *truncated mean estimators* [6, 12, 30]. Truncation is a process that consists of replacing the samples larger than some threshold value with a smaller number. This has the effect of reducing the tail of the distribution, but also of changing its expectation. Here we study the effect of replacing the values larger than some threshold b with 0, which corresponds to the new random variable $Y = X \mathbb{1}_{X \leq b}$. We consider the following classical sub-Gaussian estimator that we were not able to find in the literature: set $b = \sqrt{n\mathbb{E}[X^2]}$ and compute the empirical mean of n samples from Y . By a simple calculation, one can prove that the expectation of the removed part is at most $\mathbb{E}[X - Y] \leq \mathbb{E}[X^2]/b = \sqrt{\mathbb{E}[X^2]}/n$. Moreover, using Bernstein's inequality and the boundedness of Y , the error between the output estimate and $\mathbb{E}[Y]$ is on the order of $\sqrt{\mathbb{E}[X^2]}/n$. These two facts together imply that the overall error for estimating $\mathbb{E}[X]$ is indeed of a sub-Gaussian type. This approach can be carried out in the quantum model by performing the truncation in superposition. This is similar to what is done in previous quantum mean estimators [23, 32, 22]. In order to obtain a quantum speedup, one must balance the truncation level differently by taking $b = n\sqrt{\mathbb{E}[X^2]}$. Then, by a clever use of amplitude estimation discovered by Heinrich [23], the expectation of Y can be estimated with an error on the order of $\sqrt{\mathbb{E}[X^2]}/n$. The main drawback of this estimator is that it requires the knowledge of $\mathbb{E}[X^2]$ to perform the truncation. In previous work [23, 32, 22], the authors made further assumptions on the variance to be able to approximate b . Here, we overcome this issue by choosing the truncation level b differently. Borrowing ideas from classical estimators [30], we define b as the quantile value that satisfies $\Pr[X \geq b] = 1/n^2$. This quantile is always smaller than the previous threshold value $n\sqrt{\mathbb{E}[X^2]}$. Moreover, it can be shown that the removed part $\mathbb{E}[X - Y]$ is still on the order of $\sqrt{\mathbb{E}[X^2]}/n$. We give a new quantum algorithm for approximating this quantile with roughly n quantum experiments (Theorem 11), whereas it would require n^2 random experiments classically. Our quantile estimation algorithm builds upon the quantum minimum finding algorithm of Dürr and Høyer [19, 3] and the k th-smallest element finding algorithm of Nayak and Wu [34]. Importantly, it does not require any knowledge about $\mathbb{E}[X^2]$.

(ϵ, δ) -Approximation without side information. We follow an approach similar to that of a classical estimator described in [17]. Our algorithm (Theorem 16) uses the quantum sub-Gaussian estimator and the quantum *sequential Bernoulli estimator* described in Proposition 15. The latter estimator can estimate the mean μ of a random variable X distributed in $[0, 1]$ with constant relative error by performing $O(1/\sqrt{\mu})$ quantum experiments in expectation. The first step of the (ϵ, δ) -approximation algorithm is to compute a rough estimate $\hat{\mu}$ of μ with the sequential Bernoulli estimator. Then, the variance σ^2 of X is estimated by using again the sequential Bernoulli estimator on the random variable $(X - X')/2$ (where X' is an independent copy of X). The latter estimation is stopped if it uses more than $O(1/\sqrt{\epsilon\hat{\mu}})$ quantum experiments. We show that if $\sigma^2 \geq \Omega(\epsilon\mu)$ then the computation is not stopped and the resulting estimate $\tilde{\sigma}^2$ is close to σ^2 with high probability. Otherwise, it is stopped with high probability and we set $\tilde{\sigma} = 0$. Finally, the quantum sub-Gaussian estimator is used with the parameter $n \approx \max\left(\frac{\tilde{\sigma}}{\epsilon\mu}, \frac{1}{\sqrt{\epsilon\mu}}\right)$ to obtain a refined estimate $\tilde{\mu}$ of μ . The choice of the first

(resp. second) term in the maximum value implies that $|\tilde{\mu} - \mu| \leq \epsilon\mu$ with high probability when the variance σ^2 is larger (resp. smaller) than $\epsilon\mu$. In order to upper bound the expected number of experiments performed by this estimator, we show in Proposition 15 that the estimates $\hat{\mu}$ and $\hat{\sigma}$ obtained with the sequential Bernoulli estimator satisfy the expectation bounds $\mathbb{E}[1/\hat{\mu}] \leq 1/\mu$, $\mathbb{E}[\hat{\sigma}] \leq \sigma$ and $\mathbb{E}[1/\sqrt{\hat{\mu}}] \leq 1/\sqrt{\mu}$.

Lower bounds. We sketch the proof of optimality of the quantum sub-Gaussian estimator (Theorem 18). The lower bound is proved in the stronger quantum query model, which allows us to extend it to all the other models mentioned in Section 2.1. Our approach is inspired by the truncation level chosen in the algorithm. Given σ and n , we consider the two distributions p_0 and p_1 that output respectively $\frac{n\sigma}{\sqrt{1-1/n^2}}$ and $\frac{-n\sigma}{\sqrt{1-1/n^2}}$ with probability $1/n^2$, and 0 otherwise. The two distributions have variance σ^2 and the distance between their means is larger than $\frac{2\sigma}{n}$. Thus, any estimator that satisfies the bound $\Pr[|\tilde{\mu} - \mu| > \frac{\sigma}{n}] \leq \frac{1}{3}$ can distinguish between p_0 and p_1 with constant success probability. However, we show by a reduction to Quantum Search that it requires at least $\Omega(n)$ quantum experiments to distinguish between two distributions that differ with probability at most $1/n^2$.

2 Preliminaries

2.1 Model of input

The input to the mean estimation problem is represented by a real-valued random variable X defined on some probability space. A classical estimator accesses this input by obtaining n *i.i.d* samples of X . In this section, we describe the access model for quantum estimators and we compare it to previous models suggested in the literature. We only consider finite probability spaces for finite encoding reasons. First, we recall the definition of a random variable, and we define a classical model of access called a *random experiment*.

► **Definition 1** (RANDOM VARIABLE). A finite random variable is a function $X : \Omega \rightarrow E$ for some probability space (Ω, p) , where Ω is a finite sample set, $p : \Omega \rightarrow [0, 1]$ is a probability mass function and $E \subset \mathbb{R}$ is the support of X . As is customary, we will often omit to mention (Ω, p) when referring to the random variable X .

► **Definition 2** (RANDOM EXPERIMENT). Given a random variable X on a probability space (Ω, p) , we define a random experiment as the process of drawing a sample $\omega \in \Omega$ according to p and observing the value of $X(\omega)$.

We now introduce the concept of “q-random variable” to represent a quantum process that outputs a real number.

► **Definition 3** (Q-RANDOM VARIABLE). A q-variable is a triple (\mathcal{H}, U, M) where \mathcal{H} is a finite-dimensional Hilbert space, U is a unitary transformation on \mathcal{H} , and $M = \{M_x\}_{x \in E}$ is a projective measurement on \mathcal{H} indexed by a finite set $E \subset \mathbb{R}$. Given a random variable X on a probability space (Ω, p) , we say that a q-variable (\mathcal{H}, U, M) generates X when,

1. \mathcal{H} is a finite-dimensional Hilbert space with some basis $\{|\omega\rangle\}_{\omega \in \Omega}$ indexed by Ω .
 2. U is a unitary transformation on \mathcal{H} such that $U|0\rangle = \sum_{\omega \in \Omega} \sqrt{p(\omega)}|\omega\rangle$.
 3. $M = \{M_x\}_x$ is the projective measurement on \mathcal{H} defined by $M_x = \sum_{\omega: X(\omega)=x} |\omega\rangle\langle\omega|$.
- A random variable X is a q-random variable if it is generated by some q-variable (\mathcal{H}, U, M) .

We stress that the sample space Ω may not be known explicitly, and we do not assume that it is easy to perform a measurement in the $\{|\omega\rangle\}_{\omega \in \Omega}$ basis for instance. Often, we are given a unitary U such that $U|0\rangle = \sum_{x \in E} \sqrt{p(x)}|\psi_x\rangle|x\rangle$ for some unknown garbage unit

state $|\psi_x\rangle$, together with the measurement $M = \{I \otimes |x\rangle\langle x|\}_{x \in E}$. In this case, we can consider the q-random variable X defined on the probability space (Ω, p) where $\Omega = \{|\psi_x\rangle|x\rangle\}_{x \in E}$ and $X(|\psi_x\rangle|x\rangle) = x$.

We further assume that there exist two quantum oracles, defined below, for obtaining information on the function $X : \Omega \rightarrow E$. These two oracles can be efficiently implemented if we have access to a quantum evaluation oracle $|\omega\rangle|0\rangle \mapsto |\omega\rangle|X(\omega)\rangle$ for instance. The rotation oracle (Assumption 2) has been extensively used in previous quantum mean estimators [38, 8, 32, 22]. The comparison oracle (Assumption 1) is needed in our work to implement the quantile estimation algorithm.

► **Assumption 1** (COMPARISON ORACLE). Given a q-random variable X on a probability space (Ω, p) , and any two values $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$ such that $a < b$, there is a unitary operator $C_{a,b}$ acting on $\mathcal{H} \otimes \mathbb{C}^2$ such that for all $\omega \in \Omega$,

$$C_{a,b}(|\omega\rangle|0\rangle) = \begin{cases} |\omega\rangle|1\rangle & \text{when } a < X(\omega) \leq b, \\ |\omega\rangle|0\rangle & \text{otherwise.} \end{cases}$$

► **Assumption 2** (ROTATION ORACLE). Given a q-random variable X on a probability space (Ω, p) , and any two values $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$ such that $a < b$, there is a unitary operator $R_{a,b}$ acting on $\mathcal{H} \otimes \mathbb{C}^2$ such that for all $\omega \in \Omega$,

$$R_{a,b}(|\omega\rangle|0\rangle) = \begin{cases} |\omega\rangle \left(\sqrt{1 - \left| \frac{X(\omega)}{b} \right|} |0\rangle + \sqrt{\left| \frac{X(\omega)}{b} \right|} |1\rangle \right) & \text{when } a < X(\omega) \leq b, \\ |\omega\rangle|0\rangle & \text{otherwise.} \end{cases}$$

We now define the measure of complexity used to count the number of accesses to a q-random variable, which are referred to as *quantum experiments*.

► **Definition 4** (QUANTUM EXPERIMENT). Let X be a q-random variable that satisfies Assumptions 1 and 2. Let (\mathcal{H}, U, M) be a q-variable that generates X . We define a quantum experiment as the process of applying any of the unitaries U , $C_{a,b}$, $R_{a,b}$ (for any values of $a < b$), their inverses or their controlled versions, or performing a measurement according to M .

Note that a random experiment (Definition 2) can be simulated with two quantum experiments by computing the state $U|0\rangle$ and measuring it according to M . We briefly mention two other possible input models. First, some authors [21, 34, 23, 10, 16, 8, 29] consider the stronger query model where p is the uniform distribution and a quantum evaluation oracle is provided for the function $\omega \mapsto X(\omega)$. A second model tackles the problem of *learning from quantum states* [11, 5, 4], where the input consists of several copies of $\sum_{x \in E} \sqrt{\Pr[X = x]}|x\rangle$ (we do not have access to a unitary preparing that state). We show in Theorem 22 that no quantum speedup is achievable for our problem in the latter setting.

2.2 Tools

We will use a variant of the amplitude amplification algorithm that does not need a time parameter n as input. We call it the “sequential amplitude amplification” algorithm in reference to *sequential analysis*. The original version of this algorithm was analysed in Theorem 3 of [7, 9], with a bound on the expected complexity $\mathbb{E}[T]$. We propose a slightly different version that allows us to bound $\mathbb{E}[T^2]$ and $\mathbb{E}[1/T]$ (note that $\mathbb{E}[T] \leq \sqrt{\mathbb{E}[T^2]}$). The algorithm and its analysis are deferred to the extended version of this paper.

► **Theorem 5** (SEQUENTIAL AMPLITUDE AMPLIFICATION). *Let U be a unitary quantum algorithm and let Π be a projection operator. Define the number $p \in [0, 1]$ and the two unit states $|\psi_0\rangle, |\psi_1\rangle$ such that $U|0\rangle = \sqrt{1-p}|\psi_0\rangle + \sqrt{p}|\psi_1\rangle$ and $\Pi U|0\rangle = \sqrt{p}|\psi_1\rangle$. If $p > 0$ then the sequential amplitude amplification algorithm $\text{Seq-AAmp}(U, \Pi)$ outputs the state $|\psi_1\rangle$ with probability 1. Moreover, if we let T denote the number of applications of U , U^\dagger and $I - 2\Pi$ used by the algorithm, then $\mathbb{E}[T^2] \leq O(1/p)$ and $\mathbb{E}[1/T] \leq O(\sqrt{p})$.*

The next result is a generalization of Quantum Counting that corresponds to Theorems 11 and 12 in [9].

► **Theorem 6** (AMPLITUDE ESTIMATION, [9]). *Let U be a unitary quantum algorithm and let Π be a projection operator. Define the number $p \in [0, 1]$ such that $p = \|\Pi U|0\rangle\|^2$. Then, for any integer $n \geq 0$, the amplitude estimation algorithm $\text{AEst}(U, \Pi, n)$ outputs an amplitude estimate \tilde{p} such that, $\Pr\left[|\tilde{p} - p| \leq \frac{2\pi\sqrt{p(1-p)}}{n} + \frac{\pi^2}{n^2}\right] \geq 8/\pi^2$. The algorithm uses n applications of U , U^\dagger , $I - 2\Pi$ and $O(\log^2(n))$ 2-qubit quantum gates.*

We will also use a sequential version of the amplitude estimation algorithm that does not need a time parameter n as input. This result was first obtained by [9, Theorem 15]. We describe a variant with additional properties that is based on the sequential amplitude amplification algorithm.

► **Theorem 7** (SEQUENTIAL AMPLITUDE ESTIMATION). *There exists an algorithm, called the sequential amplitude estimation algorithm Seq-AEst , with the following properties. Let U be a unitary quantum algorithm and let Π be a projection operator. Define the number $p \in [0, 1]$ such that $p = \|\Pi U|0\rangle\|^2$. Then, the algorithm $\text{Seq-AEst}(U, \Pi)$ outputs an amplitude estimate \tilde{p} and uses a number T of applications of U , U^\dagger , $I - 2\Pi$ such that,*

1. *There is a universal constant $c \in (0, 1)$ such that $\Pr[|\tilde{p} - p| \leq cp] \geq 7/8$.*
2. *There is a universal constant c' such that $\mathbb{E}[T^2] = \mathbb{E}[1/\tilde{p}] \leq c'/p$.*
3. *There is a universal constant c'' such that $\mathbb{E}[1/T] = \mathbb{E}[\sqrt{\tilde{p}}] \leq c''\sqrt{p}$.*

Proof. The algorithm $\text{Seq-AEst}(U, \Pi)$ consists of recording the number T of applications of U , U^\dagger , $I - 2\Pi$ used by the sequential amplitude amplification algorithm $\text{Seq-AAmp}(U, \Pi)$ (Theorem 5), and choosing the estimate $\tilde{p} = 1/T^2$. The results follow immediately from Theorem 5 and Markov's inequality. ◀

3 Quantile estimation

In this section, we present a quantum algorithm for estimating the quantiles of a finite random variable X . This is a key ingredient for the sub-Gaussian estimator of Section 4. For the convenience of reading, we define a quantile in the following non-standard way (the cumulative distribution function is replaced with its complement).

► **Definition 8** (QUANTILE). *Given a discrete random variable X and a real $p \in [0, 1]$, the quantile of order p is the number $Q(p) = \sup\{x \in \mathbb{R} : \Pr[X \geq x] \geq p\}$.*

Our result is inspired by the minimum finding algorithm of Dürr and Høyer [19] and its generalization in [3]. The problem of estimating the quantiles of a set of numbers under the uniform distribution was studied before by Nayak and Wu [34, 33]. We differ from that work by allowing arbitrary distributions, and by not using the amplitude estimation algorithm. On the other hand, we restrict ourselves to finding a constant factor estimate, whereas [34, 33] can achieve any wanted accuracy.

The idea behind our algorithm is rather simple: if we compute a sequence of values $-\infty = y_0 \leq y_1 \leq y_2 \leq y_3 \leq \dots$ where each y_{j+1} is sampled from the distribution of X conditioned on $y_{j+1} \geq y_j$, then when $j \simeq \log(1/p)$ the value of y_j should be close to the quantile $Q(p)$. The complexity of sampling each y_j is on the order of $1/\Pr[X \geq y_j]$ classically, but it can be done quadratically faster in the quantum setting. We analyze a slightly different algorithm, where the sequence of samples is strictly increasing and instead of stopping after roughly $\log(1/p)$ iterations we count the number of experiments performed by the algorithm and stop when it reaches a value close to $1/\sqrt{p}$. This requires showing that the times T_j spent on sampling y_j is neither too large nor too small with high probability, which is proved in the next lemma.

► **Lemma 9.** *There is a quantum algorithm such that, given a q -random variable X and a value $x \in \mathbb{R} \cup \{-\infty, +\infty\}$, it outputs a sample y from the probability distribution of X conditioned on $y > x$. If we let T denote the number of quantum experiments performed by this algorithm, then there exist two universal constants $c_0 < c_1$ such that $\mathbb{E}[T] \leq c_1/\sqrt{\Pr[X > x]}$ and $\Pr[T < c_0/\sqrt{\Pr[X > x]}] \leq 1/10$.*

Proof. Let (\mathcal{H}, U, M) be a q -variable generating X . We use the comparison oracle $C_{x,+\infty}$ from Assumption 1 to construct the unitary $V = C_{x,+\infty}(U \otimes I)$ acting on $\mathcal{H} \otimes \mathbb{C}^2$. By definition of $C_{x,+\infty}$ and U (Section 2.1), we have that $V|0\rangle = \sum_{\omega \in \Omega: X(\omega) \leq x} \sqrt{p(\omega)}|\omega\rangle|0\rangle + \sum_{\omega \in \Omega: X(\omega) > x} \sqrt{p(\omega)}|\omega\rangle|1\rangle = \sqrt{1 - \Pr[X > x]}|\phi_0\rangle|0\rangle + \sqrt{\Pr[X > x]}|\phi_1\rangle|1\rangle$ for some unit states $|\phi_0\rangle, |\phi_1\rangle$ where $|\phi_1\rangle = \frac{1}{\sqrt{\Pr[X > x]}} \sum_{\omega: X(\omega) > x} \sqrt{p(\omega)}|\omega\rangle$. The algorithm for sampling y conditioned on $y > x$ consists of two steps. First, we use the sequential amplitude amplification algorithm **Seq-AAmp**($V, I \otimes |1\rangle\langle 1|$) from Theorem 5 on V to obtain the state $|\phi_1\rangle$. Next, we measure $|\phi_1\rangle$ according to M . The claimed properties follow directly from Theorem 5. ◀

We use the next formula for the probability that a value x occurs in the sequence $(y_j)_j$ defined before. This lemma is adapted from [19, Lemma 1].

► **Lemma 10** (Lemma 47 in [3]). *Let X be a discrete random variable. Consider the increasing sequence of random variables Y_0, Y_1, Y_2, \dots where Y_0 is a fixed value and Y_{j+1} for $j \geq 0$ is a sample drawn from X conditioned on $Y_{j+1} > Y_j$. Then, for any $x, y \in \mathbb{R}$,*

$$\Pr[x \in \{Y_1, Y_2, \dots\} \mid Y_0 = y] = \begin{cases} \Pr[X = x \mid X \geq x] & \text{when } x > y, \\ 0 & \text{otherwise.} \end{cases}$$

The quantile estimation algorithm is described in Algorithm 1 and the analysis is detailed in the extended version of this paper.

■ **Algorithm 1** Quantile estimation algorithm, $\text{Quantile}(X, p, \delta)$.

-
1. Repeat the following steps for $i = 1, 2, \dots, \lceil 6 \log(1/\delta) \rceil$.
 - a. Set $y_0 = -\infty$ and initialize a counter $C = 0$ that is incremented each time a quantum experiment is performed.
 - b. Set $j = 1$. Repeat the following process and interrupt it when $C = c'/\sqrt{p}$ (where c' is a constant chosen in the proof of Theorem 11): sample an element y_{j+1} from X conditioned on $y_{j+1} > y_j$ by using the algorithm of Lemma 9, set $j \leftarrow j + 1$.
 - c. Set $\tilde{Q}^{(i)} = y_j$.
 2. Output $\tilde{Q} = \text{median}(\tilde{Q}^{(1)}, \dots, \tilde{Q}^{(\lceil 6 \log(1/\delta) \rceil)})$.
-

► **Theorem 11** (QUANTILE ESTIMATION). *Let X be a q -random variable. Given two reals $p, \delta \in (0, 1)$, the approximate quantile \tilde{Q} produced by the quantile estimation algorithm $\text{Quantile}(X, p, \delta)$ (Algorithm 1) satisfies $Q(p) \leq \tilde{Q} \leq Q(cp)$ with probability at least $1 - \delta$, where $c < 1$ is a universal constant. The algorithm performs $O\left(\frac{\log(1/\delta)}{\sqrt{p}}\right)$ quantum experiments.*

4 Sub-Gaussian estimator

In this section, we present the main quantum algorithm for estimating the mean of a random variable with a near-quadratic speedup over the classical sub-Gaussian estimators. Our result uses the following *Bernoulli estimator*, which is a well-known adaptation of the amplitude estimation algorithm to the mean estimation problem [9, 38, 32]. The Bernoulli estimator allows us to estimate the mean of the truncated random variable $X \mathbb{1}_{a < X \leq b}$ for any a, b .

► **Proposition 12** (BERNOULLI ESTIMATOR). *There exists a quantum algorithm, called the Bernoulli estimator, with the following properties. Let X be a q -random variable and set as input a time parameter $n \geq 0$, two range values $0 \leq a < b$, and a real $\delta \in (0, 1)$ such that $n \geq \log(1/\delta)$. Then, the Bernoulli estimator $\text{BernEst}(X, n, a, b, \delta)$ outputs a mean estimate $\tilde{\mu}_{a,b}$ of $\mu_{a,b} = \mathbb{E}[X \mathbb{1}_{a < X \leq b}]$ such that $|\tilde{\mu}_{a,b} - \mu_{a,b}| \leq \frac{\sqrt{b\mu_{a,b} \log(1/\delta)}}{n} + \frac{b \log(1/\delta)^2}{n^2}$. It performs $O(n)$ quantum experiments.*

Proof. Let (\mathcal{H}, U, M) be a q -variable generating X . Using the rotation oracle $R_{a,b}$ from Assumption 2, we define the unitary algorithm $V = R_{a,b}(U \otimes I)$ acting on $\mathcal{H} \otimes \mathbb{C}^2$. In order to simplify notations, let us first assume that the random variable X is only distributed in the interval (a, b) . Then, we have $\mu = \mu_{a,b}$ and by definition of $R_{a,b}$ and U (Section 2.1) the operator V satisfies that $V|0\rangle = \sum_{\omega \in \Omega} \sqrt{p(\omega)} |\omega\rangle \left(\sqrt{1 - \frac{X(\omega)}{b}} |0\rangle + \sqrt{\frac{X(\omega)}{b}} |1\rangle \right) = \sqrt{1 - \frac{\mu}{b}} \left(\sum_{\omega \in \Omega} \sqrt{\frac{p(\omega)(b-X(\omega))}{b-\mu}} |\omega\rangle \right) |0\rangle + \sqrt{\frac{\mu}{b}} \left(\sum_{\omega \in \Omega} \sqrt{\frac{p(\omega)X(\omega)}{\mu}} |\omega\rangle \right) |1\rangle$. Thus, there exist some unit states $|\psi_0\rangle, |\psi_1\rangle$ such that $V|0\rangle = \sqrt{1 - \frac{\mu}{b}} |\psi_0\rangle + \sqrt{\frac{\mu}{b}} |\psi_1\rangle$ and $(I \otimes |1\rangle\langle 1|)V|0\rangle = \sqrt{\frac{\mu}{b}} |\psi_1\rangle$. If X takes values outside the interval (a, b) then the same result holds with $\mu_{a,b}$ in place of μ and a different definition of $|\psi_0\rangle, |\psi_1\rangle$.

Consider the output \tilde{v} of the amplitude estimation algorithm $\text{AEst}(V, \Pi, \lceil \frac{2\pi n}{\log(1/\delta)} \rceil)$ (Theorem 6) where $\Pi = I \otimes |1\rangle\langle 1|$. Then, the estimate $b\tilde{v}$ satisfies the statement of the proposition with probability $8/\pi^2$ by Theorem 6. The Bernoulli estimator consists of running $\lceil 6 \log(1/\delta) \rceil$ copies of $\text{AEst}(V, \Pi, \lceil \frac{2\pi n}{\log(1/\delta)} \rceil)$ and outputting the median of the results. The success probability is at least $1 - \delta$ by the Chernoff bound. ◀

The Bernoulli estimator can estimate the mean of a non-negative q -random variable X by setting $a = 0$ and $b = \max X$. However, its performance is worse than that of the classical sub-Gaussian estimators when the maximum of X is large compared to its variance. Our quantum sub-Gaussian estimator (Algorithm 2) uses the Bernoulli estimator in a more subtle way, and in combination with the quantile estimation algorithm.

► **Theorem 13** (SUB-GAUSSIAN ESTIMATOR). *Let X be a q -random variable with mean μ and variance σ^2 . Given a time parameter n and a real $\delta \in (0, 1)$ such that $n \geq \log(1/\delta)$, the sub-Gaussian estimator $\text{SubGaussEst}(X, n, \delta)$ (Algorithm 2) outputs a mean estimate $\tilde{\mu}$ such that, $\Pr\left[|\tilde{\mu} - \mu| \leq \frac{\sigma \log(1/\delta)}{n}\right] \geq 1 - \delta$. The algorithm performs $O(n \log^{3/2}(n) \log \log(n))$ quantum experiments.*

■ **Algorithm 2** Sub-Gaussian estimator, $\text{SubGaussEst}(X, n, \delta)$.

1. Set $k = \log n$ and $m = dn\sqrt{\log n} \frac{\log(9k/\delta)}{\log(1/\delta)}$, where $d > 1$ is a constant chosen in the proof of Theorem 13 (if k is not an integer, round n to the next power of two).
2. Compute the median η of $\lceil 30 \log(2/\delta) \rceil$ classical samples from X and define the non-negative random variables

$$Y^+ = (X - \eta) \mathbb{1}_{X \geq \eta} \quad \text{and} \quad Y^- = -(X - \eta) \mathbb{1}_{X \leq \eta}.$$

3. Compute an estimate $\tilde{\mu}_{Y^+}$ of $\mathbb{E}[Y^+]$ and an estimate $\tilde{\mu}_{Y^-}$ of $\mathbb{E}[Y^-]$ by executing the following steps with $Y := Y_+$ and $Y := Y_-$ respectively:
 - a. Compute an estimate \tilde{Q} of the quantile of order $p = \left(\frac{\log(1/\delta)}{6n}\right)^2$ of Y with failure probability $\delta/8$ by using the quantile estimation algorithm $\text{Quantile}(Y, p, \delta/8)$.
 - b. Define $a_{-1} = 0$ and $a_\ell = \frac{2^\ell}{n} \tilde{Q}$ for $\ell \geq 0$. Compute an estimate $\tilde{\mu}_\ell$ of $\mathbb{E}[Y \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}]$ with failure probability $\delta/(9k)$ for each $0 \leq \ell \leq k$, by using the Bernoulli estimator $\text{BernEst}(Y, m, a_{\ell-1}, a_\ell, \delta/(9k))$ with m quantum experiments.
 - c. Set $\tilde{\mu}_Y = \sum_{\ell=0}^k \tilde{\mu}_\ell$.
4. Output $\tilde{\mu} = \eta + \tilde{\mu}_{Y^+} - \tilde{\mu}_{Y^-}$.

Proof. First, by standard concentration inequalities, the median η computed at step 2 satisfies $|\eta - \mu| \leq 2\sigma$ with probability at least $1 - \delta/2$. Moreover, if $|\eta - \mu| \leq 2\sigma$ then $\sqrt{\mathbb{E}[(X - \eta)^2]} = \sqrt{\mathbb{E}[(X - \mu + \mu - \eta)^2]} \leq \sqrt{\mathbb{E}[(X - \mu)^2]} + |\mu - \eta| \leq 3\sigma$, by using the triangle inequality. Below we prove that for any non-negative random variable Y the estimate $\tilde{\mu}_Y$ of $\mu_Y = \mathbb{E}[Y]$ computed at step 3 satisfies

$$|\tilde{\mu}_Y - \mu_Y| \leq \frac{\sqrt{\mathbb{E}[Y^2]} \log(1/\delta)}{5n} \quad (4)$$

with probability at least $1 - \delta/4$. Using the fact that $X = \eta + Y_+ - Y_-$ and $(X - \eta)^2 = Y_+^2 + Y_-^2$, we can conclude that

$$|\tilde{\mu} - \mu| \leq \frac{\left(\sqrt{\mathbb{E}[Y_+^2]} + \sqrt{\mathbb{E}[Y_-^2]}\right) \log(1/\delta)}{5n} \leq \frac{\sqrt{2\mathbb{E}[(X - \eta)^2]} \log(1/\delta)}{5n} \leq \frac{\sigma \log(1/\delta)}{n}$$

with probability at least $1 - \delta$. The algorithm performs $O(\log(1/\delta)) \leq O(n)$ classical experiments during step 2, $O(\log(1/\delta)/\sqrt{p}) \leq O(n)$ quantum experiments during step 3.a, and $O(km) \leq O(n \log^{3/2}(n) \log \log(n))$ quantum experiments during step 3.b.

We now turn to the proof of Equation (4). We make the assumption that all the subroutines used in step 3 are successful, which is the case with probability at least $(1 - \delta/8)(1 - \delta/(9k))^{k+1} \geq 1 - \delta/4$. First, according to Theorem 11, we have $Q(p) \leq \tilde{Q} \leq Q(cp)$ for some universal constant c . It implies that $cp \leq \Pr[Y \geq Q(cp)] \leq \Pr[Y \geq \tilde{Q}] \leq \mathbb{E}[Y^2]/\tilde{Q}^2$, where the first two inequalities are by definition of the quantile function Q , and the last inequality is a standard fact. Consequently, by our choice of p ,

$$\tilde{Q} \leq \frac{6n\sqrt{\mathbb{E}[Y^2]}}{\sqrt{c} \log(1/\delta)}. \quad (5)$$

Next, we upper bound the expectation of the part of Y that is above the largest threshold $a_k = \tilde{Q}$ considered in step 3.b. By Cauchy-Schwarz' inequality, we have $\mathbb{E}[Y \mathbb{1}_{Y > \tilde{Q}}] \leq$

$\sqrt{\mathbb{E}[Y^2] \Pr[Y > \tilde{Q}]}$. Moreover, by definition of Q , $\Pr[Y > \tilde{Q}] \leq \Pr[Y > Q(p)] \leq p$. Thus,

$$\mathbb{E}[Y \mathbb{1}_{Y > \tilde{Q}}] \leq \frac{\sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{6n}. \quad (6)$$

The expectation of Y is decomposed into the sum $\mu_Y = \sum_{\ell=0}^k \mu_\ell + \mathbb{E}[Y \mathbb{1}_{Y > a_k}]$, where $\mu_\ell = \mathbb{E}[Y \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}]$ is estimated at step 3.b. We have $|\tilde{\mu}_\ell - \mu_\ell| \leq \frac{\sqrt{a_\ell \mu_\ell} \log(1/\delta)}{dn \sqrt{\log n}} + \frac{a_\ell \log(1/\delta)^2}{d^2 n^2 \log n}$ for all $0 \leq \ell \leq k$ according to Proposition 12. Thus, by the triangle inequality,

$$\begin{aligned} |\tilde{\mu}_Y - \mu_Y| &\leq \sum_{\ell=0}^k |\tilde{\mu}_\ell - \mu_\ell| + \mathbb{E}[Y \mathbb{1}_{Y > a_k}] \\ &\leq \sum_{\ell=0}^k \frac{\sqrt{a_\ell \mu_\ell} \log(1/\delta)}{dn \sqrt{\log n}} + \sum_{\ell=0}^k \frac{a_\ell \log(1/\delta)^2}{d^2 n^2 \log n} + \mathbb{E}[Y \mathbb{1}_{Y > a_k}] \\ &\leq \frac{\tilde{Q} \log(1/\delta)}{dn^2 \sqrt{\log n}} + \sum_{\ell=1}^k \frac{\sqrt{2\mathbb{E}[Y^2 \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}] \log(1/\delta)}}{dn \sqrt{\log n}} + \frac{2\tilde{Q} \log(1/\delta)^2}{d^2 n^2 \log n} + \mathbb{E}[Y \mathbb{1}_{Y > a_k}] \\ &\leq \frac{\sqrt{2k} \sqrt{\sum_{\ell=1}^k \mathbb{E}[Y^2 \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}] \log(1/\delta)}}{dn \sqrt{\log n}} + \frac{3\tilde{Q} \log(1/\delta)^2}{dn^2 \sqrt{\log n}} + \mathbb{E}[Y \mathbb{1}_{Y > a_k}] \\ &\leq \frac{\sqrt{2k} \sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{dn \sqrt{\log n}} + \frac{3\tilde{Q} \log(1/\delta)^2}{dn^2 \sqrt{\log n}} + \mathbb{E}[Y \mathbb{1}_{Y > a_k}] \\ &\leq \frac{\sqrt{2} \sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{dn} + \frac{18 \sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{\sqrt{cdn} \sqrt{\log n}} + \frac{\sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{6n} \\ &\leq \frac{\sqrt{\mathbb{E}[Y^2] \log(1/\delta)}}{5n} \end{aligned}$$

where the third step uses $a_0 \mu_0 \leq a_0^2 = (\tilde{Q}/n)^2$ and $a_\ell \mu_\ell \leq (a_\ell/a_{\ell-1}) \mathbb{E}[Y^2 \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}] \leq 2\mathbb{E}[Y^2 \mathbb{1}_{a_{\ell-1} < Y \leq a_\ell}]$ when $\ell \geq 1$, the fourth step uses the Cauchy-Schwarz inequality, the sixth step uses Equations (5) and (6), and in the last step we choose $d = 600/\sqrt{c}$. ◀

5 (ϵ, δ) -Estimators

We study the (ϵ, δ) -approximation problem under two different scenarios. First, we consider the case where we know an upper bound Δ on the coefficient of variation $|\sigma/\mu|$. As a direct consequence of Theorem 13 we obtain the following estimator that subsumes a similar result shown in [22] for non-negative random variables.

► **Corollary 14** (Relative estimator). *There exists a quantum algorithm with the following properties. Let X be a q -random variable with mean μ and variance σ^2 , and set as input a value $\Delta \geq |\sigma/\mu|$ and two reals $\epsilon, \delta \in (0, 1)$. Then, the algorithm outputs a mean estimate $\tilde{\mu}$ such that $\Pr[|\tilde{\mu} - \mu| > \epsilon|\mu|] \leq \delta$ and it performs $\tilde{O}(\frac{\Delta}{\epsilon} \log(1/\delta))$ quantum experiments.*

Proof. The algorithm runs the sub-Gaussian estimator $\text{SubGaussEst}(X, \frac{\Delta}{\epsilon} \log(1/\delta), \delta)$. ◀

Next, we construct a parameter-free estimator that performs $\tilde{O}((\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}) \log(1/\delta))$ quantum experiments in expectation for any random variable distributed in $[0, 1]$. We follow an approach similar to the classical \mathcal{AA} algorithm described in [17]. We first give a sequential estimator that approximates the mean with constant relative error and that performs $O(1/\sqrt{\mu})$ quantum experiments in expectation. We use the term “sequential” in reference to sequential analysis techniques. The classical counterpart of this estimator is the Stopping Rule Algorithm in [17].

► **Proposition 15** (SEQUENTIAL BERNOULLI ESTIMATOR). *There is an algorithm, called the sequential Bernoulli estimator, with the following properties. Let X be a q -random variable distributed in $[0, 1]$ with mean μ . Then, the sequential Bernoulli estimator $\text{Seq-BernEst}(X)$ outputs an estimate $\tilde{\mu}$ and performs a number T of quantum experiments such that,*

1. *There is a universal constant $c \in (0, 1)$ such that $\Pr[|\tilde{\mu} - \mu| \leq c\mu] \geq 7/8$.*
2. *There is a universal constant c' such that $\mathbb{E}[T^2] = \mathbb{E}[1/\tilde{\mu}] \leq c'/\mu$.*
3. *There is a universal constant c'' such that $\mathbb{E}[\sqrt{\tilde{\mu}}] \leq c''\sqrt{\mu}$.*

Proof. The algorithm is identical to the one of Proposition 12 with $a = 0$ and $b = 1$, except that the amplitude estimation algorithm is replaced with the sequential amplitude estimation algorithm (Theorem 7). The algorithm inherits the properties proved in Theorem 7. ◀

The expected number of experiments performed by the sequential Bernoulli estimator is $\mathbb{E}[T] \leq \sqrt{\mathbb{E}[T^2]} \leq 1/\sqrt{\mu}$. The output $\tilde{\mu}$ of the sequential Bernoulli estimator can be used in the Bernoulli estimator (Proposition 12) with parameter $n = 8 \log(1/\delta)/(\epsilon\sqrt{\tilde{\mu}})$ to solve the (ϵ, δ) -approximation problem. However, the expected number of experiments performed with this approach is $O(\log(1/\delta)/(\epsilon\sqrt{\mu}))$. We propose a better algorithm with an improved dependence on ϵ . The algorithm uses the sequential Bernoulli estimator and the sub-Gaussian estimator.

■ **Algorithm 3** Sequential (ϵ, δ) -estimator.

1. For $i = 1, \dots, 32 \log(1/\delta)$:
 - a. Compute an estimate $\tilde{\mu}_X$ of $\mu = \mathbb{E}[X]$ by using the sequential Bernoulli estimator $\text{Seq-BernEst}(X)$ (Proposition 15).
 - b. Let Y denote the random variable $(X - X')^2/2$ where X' is independent from X and identically distributed. Compute an estimate $\tilde{\mu}_Y$ of $\mu_Y = \mathbb{E}[Y]$ by using the sequential Bernoulli estimator $\text{Seq-BernEst}(Y)$ (Proposition 15). Stop the computation if it performs more than $\frac{c_1}{\sqrt{\epsilon\mu_X}}$ quantum experiments (where c_1 is a constant chosen in the proof of Theorem 16) and set $\tilde{\mu}_Y = 0$.
 - c. Compute a second estimate $\tilde{\mu}_X^{(i)}$ of μ by using the sub-Gaussian estimator $\text{SubGaussEst}(X, n, 15/16)$ (Theorem 13) with $n = c_2 \max\left(\frac{\sqrt{\tilde{\mu}_Y}}{\epsilon\mu_X}, \frac{1}{\sqrt{\epsilon\mu_X}}\right)$ (where c_2 is a constant chosen in the proof of Theorem 16).
2. Output $\tilde{\mu} = \text{median}\left(\tilde{\mu}_X^{(1)}, \dots, \tilde{\mu}_X^{(32 \log(1/\delta))}\right)$.

► **Theorem 16** (SEQUENTIAL RELATIVE ESTIMATOR). *Let X be a q -random variable distributed in $[0, 1]$ with mean μ and variance σ^2 . Given two reals $\epsilon, \delta \in (0, 1)$ the estimate $\tilde{\mu}$ output by the sequential relative estimator (Algorithm 3) satisfies $\Pr[|\tilde{\mu} - \mu| > \epsilon\mu] \leq \delta$. The algorithm performs $\tilde{O}\left(\left(\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}\right) \log(1/\delta)\right)$ quantum experiments in expectation.*

Proof. We prove that, for a fixed value of i , the estimate $\tilde{\mu}_X^{(i)}$ computed at step 1.c satisfies $\Pr[|\tilde{\mu}_X^{(i)} - \mu| \leq \epsilon\mu] \geq 5/8$ and the number of experiments performed during its computation is $\tilde{O}\left(\left(\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}\right)\right)$ in expectation. The theorem follows by the Chernoff bound and the linearity of expectation.

Let c, c', c'' denote the constants mentioned in Proposition 15, and set $c_1 = 16c'\sqrt{1+c}$ and $c_2 = 4(1+c)/\sqrt{1-c}$. We assume that $|\tilde{\mu}_X - \mu| \leq c\mu$ at step 1.a, which is the case with probability at least $7/8$ by Proposition 15. The analysis of steps 1.b and 1.c is split into two cases to show that $\Pr[|\tilde{\mu}_X^{(i)} - \mu| \leq \epsilon\mu] \geq 5/8$. First, if $\sigma \leq \sqrt{\epsilon\mu}$, then we can

ignore step 1.b and consider the second term in the max at step 1.c. By Theorem 13, the estimate $\tilde{\mu}_X^{(i)}$ satisfies $|\tilde{\mu}_X^{(i)} - \mu| \leq \frac{4\sigma}{c_2/\sqrt{\epsilon\mu_X}} \leq \frac{4\sqrt{1+c}}{c_2}\epsilon\mu \leq \epsilon\mu$ with probability 15/16.

Secondly, if $\sigma \geq \sqrt{\epsilon\mu}$, then by Proposition 15 and the fact that $\mu_Y = \sigma^2$, the estimate $\tilde{\mu}_Y$ computed at step 1.b satisfies $|\tilde{\mu}_Y - \sigma^2| \leq c\sigma^2$ with probability 7/8 if we remove the stopping condition. Since we assumed that $\tilde{\mu}_X \leq (1+c)\mu$, the computation is interrupted if it performs more than $\frac{c_1}{\sqrt{\epsilon\mu_X}} \geq \frac{c_1}{\sqrt{(1+c)\mu_Y}} = \frac{16c'}{\sqrt{\mu_Y}}$ experiments. However, by Proposition 15 and Markov's inequality, the number of experiments performed by the sequential Bernoulli estimator at step 1.b is at most $16c'/\sqrt{\mu_Y}$ with probability at least 15/16. Consequently, we can assume that $\tilde{\mu}_Y \geq (1-c)\sigma^2$ with success probability at least $7/8 \cdot 15/16$. In this case, by considering the first term in the max at step 1.c, the estimate $\tilde{\mu}_X^{(i)}$ satisfies $|\tilde{\mu}_X^{(i)} - \mu| \leq \frac{4\sigma}{c_2\sqrt{\mu_Y/(\epsilon\mu_X)}} \leq \frac{4(1+c)}{c_2\sqrt{1-c}}\epsilon\mu \leq \epsilon\mu$ with probability 15/16. The overall success probability is at least $(7/8)^2(15/16)^2 \geq 5/8$.

We now analyse the expected number of quantum experiments performed during the computation of $\tilde{\mu}_X^{(i)}$. Step 1.a performs $O(1/\sqrt{\mu})$ experiments in expectation by Proposition 15. Step 1.b is stopped after $O(1/(\sqrt{\epsilon\mu}))$ experiments in expectation since $\mathbb{E}[1/\sqrt{\tilde{\mu}_X}] \leq O(1/\sqrt{\mu})$ by Proposition 15. Step 1.c performs $\tilde{O}\left(\max\left(\frac{\sqrt{\mu_Y}}{\epsilon\mu_X}, \frac{1}{\sqrt{\epsilon\mu_X}}\right)\right)$ experiments by Theorem 13. The estimates $\tilde{\mu}_Y$ and $\tilde{\mu}_X$ are independent if we ignore the stopping condition at step 1.b, in which case $\mathbb{E}\left[\frac{\sqrt{\mu_Y}}{\mu_X}\right] = \mathbb{E}\left[\frac{1}{\mu_X}\right]\mathbb{E}[\sqrt{\mu_Y}] \leq O\left(\frac{\sigma}{\mu}\right)$ by Proposition 15. The stopping condition can only decrease this quantity. Thus, step 1.c performs $\tilde{O}\left(\max\left(\frac{\sigma}{\epsilon\mu}, \frac{1}{\sqrt{\epsilon\mu}}\right)\right)$ experiments in expectation. \blacktriangleleft

6 Lower bounds

We prove several lower bounds for the mean estimation problem under different scenarios. In Section 6.1, we study the number of experiments that must be performed to estimate the mean with a sub-Gaussian error rate. In Section 6.2, we study the number of experiments needed to solve the (ϵ, δ) -approximation problem. Finally, in Section 6.3, we consider the mean estimation problem in the state-based model, where the input consists of several copies of a quantum state encoding a distribution.

6.1 Sub-Gaussian estimation

We show that the quantum sub-Gaussian estimator described in Theorem 13 is optimal up to a polylogarithmic factor. We make use of the following lower bound for Quantum Search in the small-error regime.

► **Proposition 17** (Theorem 4 in [13]). *Let $N > 0$, $1 \leq K \leq 0.9N$ and $\delta \geq 2^{-N}$. Let $T(N, K, \delta)$ be the minimum number of quantum queries any algorithm must use to decide with failure probability at most δ whether a function $f : [N] \rightarrow \{0, 1\}$ has 0 or K preimages of 1. Then, $T(N, K, \delta) \geq \Omega(\sqrt{N/K} \log(1/\delta))$.*

We construct two particular probability distributions that allow us to reduce the Quantum Search problem to the sub-Gaussian mean estimation problem.

► **Theorem 18.** *Let $n > 1$ and $\delta \in (0, 1)$ such that $n \geq 2 \log(1/\delta)$. Fix $\sigma > 0$ and consider the family \mathcal{P}_σ of all q -random variables with variance σ^2 . Let $T(n, \sigma, \delta)$ be the minimum number of quantum experiments any algorithm must perform to compute with failure probability at most δ a mean estimate $\tilde{\mu}$ such that $|\tilde{\mu} - \mu| \leq \frac{\sigma \log(1/\delta)}{n}$ for any $X \in \mathcal{P}_\sigma$ with mean μ . Then, $T(n, \sigma, \delta) \geq \Omega(n)$.*

Proof. Let $m = \frac{n}{\log(1/\delta)}$ and $b = \frac{m}{\sqrt{1-1/m^2}}\sigma$. We define the probability distribution p_0 with support $\{0, b\}$ that takes value b with probability $\frac{1}{m^2}$. Similarly, we define the probability distribution p_1 with support $\{0, -b\}$ that takes value $-b$ with probability $\frac{1}{m^2}$. The variance of each distribution is equal to σ^2 . Moreover, the means μ_0 and μ_1 of the two distributions satisfy that,

$$\mu_0 - \mu_1 > 2 \frac{\sigma \log(1/\delta)}{n}. \quad (7)$$

Let N, K be two integers such that $N \geq \log(1/\delta)$ and $K/N = 1/m^2$ (assuming m is rational). Let F_0 be the family of all functions $f : [N] \rightarrow \{0, 1\}$ with exactly K preimages of 1. Similarly, let F_1 be the family of all functions $f : [N] \rightarrow \{-1, 0\}$ with exactly K preimages of -1 . By using Proposition 17, it is easy to see that any algorithm that can distinguish between $f \in F_0$ and $f \in F_1$ with success probability $1 - \delta$ must use at least $\Omega(\sqrt{N/K} \log(1/\delta)) = \Omega(m \log(1/\delta)) = \Omega(n)$ quantum queries to f . We associate with each function $f \in F_0 \cup F_1$ the q-variable $(\mathcal{H}, U, M)_f$ where $\mathcal{H} = \mathbb{C}^{N+2}$, $U|0\rangle = \frac{1}{\sqrt{N}} \sum_{x \in [N]} |x\rangle |f(x)\rangle$, and $M = \{I \otimes |0\rangle\langle 0|, I \otimes |-1\rangle\langle -1|, I \otimes |1\rangle\langle 1|\}$. The random variable X generated by $(\mathcal{H}, U, M)_f$ is distributed according to p_0 if $f \in F_0$, and according to p_1 if $f \in F_1$. Moreover, one quantum experiment with respect to X can be simulated with one quantum query to f . Consequently, any algorithm that can distinguish between a random variable distributed according to p_0 or p_1 with success probability $1 - \delta$ must perform at least $\Omega(n)$ quantum experiments. On the other hand, by Equation (7), if an algorithm can estimate the mean with an error rate smaller than $\frac{\sigma \log(1/\delta)}{n}$ then it can distinguish between $f \in F_0$ and $f \in F_1$. Thus, $T(n, \sigma, \delta) \geq \Omega(n)$. ◀

6.2 (ϵ, δ) -Estimation

We consider the (ϵ, δ) -estimation problem in the parameter-free setting, when the coefficient of variation is unknown. We make use of the next lower bound for Quantum Counting.

► **Proposition 19** (Theorem 4.2.6 in [33]). *Let $N > 0$, $1 < K \leq N$ and $\epsilon \in (\frac{1}{4K}, 1)$. Consider the set of all quantum algorithms such that, given a query oracle to any function $f : [N] \rightarrow \{0, 1\}$, they return an estimate \tilde{C} of the number C of preimages of 1 in f such that $|\tilde{C} - C| \leq \epsilon C$ with probability at least $2/3$. Let $T_K(N, \epsilon)$ be the minimum number of quantum queries any such algorithm must use when the oracle has exactly K preimages of 1. Then, $T_K(N, \epsilon) \geq \Omega\left(\frac{\sqrt{K(N-K)}}{\epsilon K+1} + \sqrt{\frac{N}{\epsilon K+1}}\right)$.*

We obtain by a simple reduction to the above problem that the result described in Theorem 16 is nearly optimal.

► **Proposition 20.** *Let $\epsilon \in (0, 1)$. Let $\mathcal{P}_{\mathcal{B}}$ denote the family of all q-random variables that follow a Bernoulli distribution. Consider any algorithm that takes as input $X \in \mathcal{P}_{\mathcal{B}}$ and that outputs a mean estimate $\tilde{\mu}$ such that $|\tilde{\mu} - \mathbb{E}[X]| \leq \epsilon \mathbb{E}[X]$ with probability at least $2/3$. Then, for any $\mu \in (0, 1)$, there exists $X \in \mathcal{P}_{\mathcal{B}}$ with mean μ such that the algorithm performs at least $\Omega\left(\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}\right)$ quantum experiments on input X , where $\sigma^2 = \text{Var}[X]$.*

Proof. Given $\epsilon \in (0, 1)$ and $\mu \in (0, 1)$, we choose two integers K and N such that $K > 1/(4\epsilon)$ and $K/N = \mu$ (assuming μ is rational). Similarly to the proof of Theorem 18, we associate with each function $f : [N] \rightarrow \{0, 1\}$ the q-variable $(\mathcal{H}, U, M)_f$ where $\mathcal{H} = \mathbb{C}^{N+2}$, $U|0\rangle = \frac{1}{\sqrt{N}} \sum_{x \in [N]} |x\rangle |f(x)\rangle$, and $M = \{I \otimes |0\rangle\langle 0|, I \otimes |1\rangle\langle 1|\}$. If an algorithm can estimate the mean

of any Bernoulli random variable with error ϵ and success probability $2/3$, then it can be used to count the number of preimages of 1 in f with the same accuracy. Thus, by Proposition 19, it must perform at least $\Omega\left(\frac{\sqrt{K(N-K)}}{\epsilon K+1} + \sqrt{\frac{N}{\epsilon K+1}}\right) = \Omega\left(\frac{\sqrt{\mu(1-\mu)}}{\epsilon\mu+1/N} + \frac{1}{\sqrt{\epsilon\mu+1/N}}\right) = \Omega\left(\frac{\sigma}{\epsilon\mu} + \frac{1}{\sqrt{\epsilon\mu}}\right)$ quantum experiments on a q -random variable with mean μ and variance $\sigma^2 = \mu(1-\mu)$. ◀

6.3 State-based estimation

We consider the *state-based* model where the input consists of several copies of a quantum state $|p\rangle = \sum_{x \in E} \sqrt{p(x)}|x\rangle$ encoding a distribution p over E . This model is weaker than the one described before, since it does not provide access to a unitary algorithm preparing $|p\rangle$. We prove that no quantum speedup is achievable in this setting. Our result uses the next lower bound on the number of copies needed to distinguish two states.

► **Lemma 21.** *Let $\delta \in (0, 1)$ and consider two probability distributions p_0 and p_1 with the same finite support $E \subset \mathbb{R}$. Define the states $|\phi_0\rangle = \sum_{x \in E} \sqrt{p_0(x)}|x\rangle$ and $|\phi_1\rangle = \sum_{x \in E} \sqrt{p_1(x)}|x\rangle$. Then, the smallest integer T such that there is an algorithm that can distinguish $|\phi_0\rangle^{\otimes T}$ from $|\phi_1\rangle^{\otimes T}$ with success probability at least $1 - \delta$ satisfies $T \geq \frac{\ln(1/(4\delta))}{D(p_0\|p_1)}$, where $D(p_0\|p_1) = \sum_{x \in E} p_0(x) \ln\left(\frac{p_0(x)}{p_1(x)}\right)$ is the KL-divergence from p_0 to p_1 .*

Proof. According to Helstrom's bound [25] the best success probability to distinguish between two states $|\phi\rangle$ and $|\phi'\rangle$ is $\frac{1}{2}(1 + \sqrt{1 - |\langle\phi|\phi'\rangle|^2})$. Thus, the smallest number T needed to distinguish $|\phi_0\rangle^{\otimes T}$ from $|\phi_1\rangle^{\otimes T}$ must satisfy $\frac{1}{2}(1 + \sqrt{1 - \langle\phi_0|\phi_1\rangle^{2T}}) \geq 1 - \delta$. It implies that $T \geq \frac{-\ln(1 - (1 - 2\delta)^2)}{-2 \ln(\langle\phi_0|\phi_1\rangle)} \geq \frac{\ln(1/(4\delta))}{-2 \ln\left(\sum_x p_0(x) \sqrt{\frac{p_1(x)}{p_0(x)}}\right)} \geq \frac{\ln(1/(4\delta))}{\sum_x p_0(x) \ln\left(\frac{p_0(x)}{p_1(x)}\right)} = \frac{\ln(1/(4\delta))}{D(p_0\|p_1)}$ where the second inequality uses the concavity of the logarithm function. ◀

We use the above lemma to show that no quantum mean estimator can perform better than the classical sub-Gaussian estimators in the state-based input model.

► **Theorem 22.** *Let $n > 1$ and $\delta \in (0, 1)$ such that $n \geq 2 \log(1/\delta)$. Fix $\sigma > 0$ and consider the family \mathcal{P}_σ of all distributions with finite support whose variance lies in the interval $[\sigma^2, 4\sigma^2]$. For any $p \in \mathcal{P}_\sigma$ with support $E \subset \mathbb{R}$, define the state $|p\rangle = \sum_{x \in E} \sqrt{p(x)}|x\rangle$. Let $T(n, \sigma, \delta)$ be the smallest integer such that there exists an algorithm that receives the state $|p\rangle^{\otimes T(n, \sigma, \delta)}$ for any $p \in \mathcal{P}_\sigma$, and that outputs an estimate $\tilde{\mu}$ of the mean μ of p such that $\Pr\left[|\tilde{\mu} - \mu| > \sqrt{\frac{\sigma^2 \log(1/\delta)}{n}}\right] \leq \delta$. Then, $T(n, \sigma, \delta) \geq \Omega(n)$.*


Proof. Let $m = \frac{n}{\log(1/\delta)}$, $b = \frac{m}{\sqrt{m-1}}\sigma$ and $\alpha = 2 \ln\left(1 + \sqrt{1 - \frac{1}{m}}\right)$. We define the two distributions p_0 and p_1 with support $E = \{0, b\}$ such that $p_0(b) = \frac{e^\alpha}{m}$ and $p_1(b) = \frac{1}{m}$. Let μ_0 and σ_0^2 (resp. μ_1 and σ_1^2) denote the expectation and the variance of p_0 (resp. p_1). Observe that $p_0, p_1 \in \mathcal{P}_\sigma$ since $\sigma_0 \in [\sigma, 2\sigma]$ and $\sigma_1 = \sigma$. Moreover, $\mu_0 - \mu_1 = \sigma \frac{e^\alpha - 1}{\sqrt{m-1}} = \sigma(e^{\alpha/2} + 1) \frac{e^{\alpha/2} - 1}{\sqrt{m-1}} > 2\sqrt{\frac{\sigma^2 \log(1/\delta)}{n}}$. Thus, we can distinguish $|p_0\rangle^{\otimes T(n, \sigma, \delta)}$ from $|p_1\rangle^{\otimes T(n, \sigma, \delta)}$ with failure probability δ by using any optimal algorithm that satisfies the error bound stated in the theorem. Since the KL-divergence from p_0 to p_1 is $D(p_0\|p_1) \leq p_0(b) \ln\left(\frac{p_0(b)}{p_1(b)}\right) = \frac{\alpha e^\alpha}{m^2} \leq \frac{6}{m}$, we must have $T(n, \sigma, \delta) \geq \Omega\left(\frac{\log(1/\delta)}{D(p_1\|p_0)}\right) = \Omega(n)$ by Lemma 21. ◀

References

- 1 D. S. Abrams and C. P. Williams. Fast quantum algorithms for numerical integrals and stochastic processes, 1999. [arXiv:quant-ph/9908083](#).
- 2 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.
- 3 J. van Apeldoorn, A. Gilyén, S. Gribling, and R. de Wolf. Quantum SDP-solvers: Better upper and lower bounds. *Quantum*, 4:230, 2020.
- 4 S. Arunachalam, A. Belovs, A. M. Childs, R. Kothari, A. Rosmanis, and R. de Wolf. Quantum coupon collector. In *Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, pages 10:1–10:17, 2020.
- 5 S. Arunachalam and R. de Wolf. Optimal quantum sample complexity of learning algorithms. *Journal of Machine Learning Research*, 19(1):2879–2878, 2018.
- 6 P. J. Bickel. On some robust estimates of location. *The Annals of Mathematical Statistics*, 36(3):847–858, 1965.
- 7 M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.
- 8 G. Brassard, F. Dupuis, S. Gambs, and A. Tapp. An optimal quantum algorithm to approximate the mean and its application for approximating the median of a set of points over an arbitrary distance, 2011. [arXiv:1106.4267 \[quant-ph\]](#).
- 9 G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.
- 10 S. Bravyi, A. W. Harrow, and A. Hassidim. Quantum algorithms for testing properties of distributions. *IEEE Transactions on Information Theory*, 57(6):3971–3981, 2011.
- 11 N. H. Bshouty and J. C. Jackson. Learning dnf over the uniform distribution using a quantum example oracle. *SIAM Journal on Computing*, 28(3):1136–1153, 1999.
- 12 S. Bubeck, N. Cesa-Bianchi, and G. Lugosi. Bandits with heavy tail. *IEEE Transactions on Information Theory*, 59(11):7711–7717, 2013.
- 13 H. Buhrman, R. Cleve, R. de Wolf, and C. Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 358–368, 1999.
- 14 R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995.
- 15 O. Catoni. Challenging the empirical mean and empirical variance: A deviation study. *Annales de l’Institut Henri Poincaré, Probabilités et Statistiques*, 48(4):1148–1185, 2012.
- 16 S. Chakraborty, E. Fischer, A. Matsliah, and R. de Wolf. New results on quantum property testing. In *Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 145–156, 2010.
- 17 P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing*, 29(5):1484–1496, 2000.
- 18 L. Devroye, M. Lerasle, G. Lugosi, and R. I. Oliveira. Sub-Gaussian mean estimators. *The Annals of Statistics*, 44(6):2695–2725, 2016.
- 19 C. Dürr and P. Høyer. A quantum algorithm for finding the minimum, 1996. [arXiv:quant-ph/9607014](#).
- 20 L. Gajek, W. Niemiro, and P. Pokarowski. Optimal Monte Carlo integration with fixed relative precision. *Journal of Complexity*, 29(1):4–26, 2013.
- 21 L. K. Grover. A framework for fast quantum mechanical algorithms. In *Proceedings of the 30th Symposium on Theory of Computing (STOC)*, pages 53–62, 1998.
- 22 Y. Hamoudi and F. Magniez. Quantum Chebyshev’s inequality and applications. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 69:1–69:16, 2019.
- 23 S. Heinrich. Quantum summation with an application to integration. *Journal of Complexity*, 18(1):1–50, 2002.

- 24 S. Heinrich. From Monte Carlo to quantum computation. *Mathematics and Computers in Simulation*, 62(3–6):219–230, 2003.
- 25 C. W. Helstrom. Quantum detection and estimation theory. *Journal of Statistical Physics*, 1(2):231–252, 1969.
- 26 M. Huber. An optimal (ϵ, δ) -randomized approximation scheme for the mean of random variables with bounded relative variance. *Random Structures & Algorithms*, 55(2):356–370, 2019.
- 27 M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.
- 28 J. C. H. Lee and P. Valiant. Optimal sub-gaussian mean estimation in \mathbb{R} , 2020. [arXiv:2011.08384 \[math.ST\]](#).
- 29 T. Li and X. Wu. Quantum query complexity of entropy estimation. *IEEE Transactions on Information Theory*, 65(5):2899–2921, 2019.
- 30 G. Lugosi and S. Mendelson. Mean estimation and regression under heavy-tailed distributions: A survey. *Foundations of Computational Mathematics*, 19(5):1145–1190, 2019.
- 31 V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 672–679, 2008.
- 32 A. Montanaro. Quantum speedup of Monte Carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301, 2015.
- 33 A. Nayak. *Lower Bounds for Quantum Computation and Communication*. PhD thesis, University of California, Berkeley, 1999.
- 34 A. Nayak and F. Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the 31st Symposium on Theory of Computing (STOC)*, pages 384–393, 1999.
- 35 A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley & Sons, 1983.
- 36 E. Novak. Quantum complexity of integration. *Journal of Complexity*, 17(1):2–16, 2001.
- 37 L. J. Schulman and V. V. Vazirani. A computationally motivated definition of parametric estimation and its applications to the Gaussian distribution. *Combinatorica*, 25(4):465–486, 2005.
- 38 B. M. Terhal. *Quantum Algorithms and Quantum Entanglement*. PhD thesis, University of Amsterdam, 1999.
- 39 J. F. Traub and H. Wozniakowski. Path integration on a quantum computer. *Quantum Information Processing*, 1(5):365–388, 2002.

Improved Approximation Algorithms for Tverberg Partitions

Sariel Har-Peled 

Department of Computer Science, University of Illinois, Urbana, IL, USA

Timothy Zhou

Department of Computer Science, University of Illinois, Urbana, IL, USA

Abstract

Tverberg's theorem states that a set of n points in \mathbb{R}^d can be partitioned into $\lceil n/(d+1) \rceil$ sets whose convex hulls all intersect. A point in the intersection (aka Tverberg point) is a centerpoint, or high-dimensional median, of the input point set. While randomized algorithms exist to find centerpoints with some failure probability, a partition for a Tverberg point provides a certificate of its correctness.

Unfortunately, known algorithms for computing exact Tverberg points take $n^{O(d^2)}$ time. We provide several new approximation algorithms for this problem, which improve running time or approximation quality over previous work. In particular, we provide the first strongly polynomial (in both n and d) approximation algorithm for finding a Tverberg point.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Geometric spanners, vertex failures, robustness

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.51

Related Version Full Version: <https://arxiv.org/abs/2007.08717> [17]

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF award CCF-1907400.

Timothy Zhou: Work on this paper was partially supported by a NSF AF award CCF-1907400.

Acknowledgements The authors thank Timothy Chan, Wolfgang Mulzer, David Rolnick, and Pablo Soberon-Bravo for providing useful references.

1 Introduction

Given a set P of n points in the plane and a query point q , classification problems ask whether q belongs to the same class as P . Some algorithms use the convex hull $\mathcal{CH}(P)$ as a decision boundary for classifying q . However, in realistic datasets, P may be noisy and contain outliers, and even one faraway point can dramatically enlarge the hull of P . Thus, we would like to measure how deeply q lies within P in way that is more robust against noise.

In this paper, we investigate the notion of Tverberg depth. However, there are many related measures of depth in the literature, including:

1. **Tukey depth.** The Tukey depth of q is the minimum number of points that must be removed before q becomes a vertex of the convex hull. Computing the depth is equivalent to computing the closed halfspace that contains q and the smallest number of points of P , and this takes $O(n \log n)$ time in the plane [7].
2. **Centerpoint.** In \mathbb{R}^d , a point with Tukey depth $n\alpha$ is an α -centerpoint. There is always a $1/(d+1)$ -centerpoint, known simply as the centerpoint, which can be computed exactly in $O(n^{d-1})$ time [19, 7]. It can be approximated using the centerpoint of a sample [11], but getting a polynomial-time (in both n and d) approximation algorithm proved challenging. Clarkson *et al.* [11] provided an algorithm that computes a $1/4d^2$ -centerpoint in roughly



© Sariel Har-Peled and Timothy Zhou;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 51; pp. 51:1–51:15

Leibniz International Proceedings in Informatics



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

- $O(d^9)$ time. Miller and Sheehy [23] derandomized it to find a (roughly) $1/2d^2$ -centerpoint in $n^{O(\log d)}$ time. More recently, Har-Peled and Mitchell [16] improved the running time to compute a (roughly) $1/d^2$ -centerpoint in (roughly) $O(d^7)$ time.
3. **Onion depth.** Imagine peeling away the vertices of the current convex hull and removing them from P . The onion depth is the number of layers which must be removed before the point q is exposed. The convex layers of points in the plane can be computed in $O(n \log n)$ time by an algorithm of Chazelle [8]. The structure of convex layers is well-understood for random points [12] and grid points [13].
 4. **Uncertainty.** Another model considers uncertainty about the locations of the points. Suppose that each point of P has a certain probability of existing, or alternatively, its location is given via a distribution. The depth of query point q is the probability that q is in the convex hull once P has been sampled. Under certain assumptions, this probability can be computed exactly in $O(n \log n)$ time [2]. Unfortunately, the computed value might be very close to zero or one, and therefore tricky to interpret.
 5. **Simplicial depth.** The simplicial depth of q is the number of simplices induced by P containing it. This number can be approximated quickly after some preprocessing [1]. However, it can be quite large for a point which is intuitively shallow.

Tverberg depth

Given a set P of n points in \mathbb{R}^d , a **Tverberg partition** is a partition of P into k disjoint sets P_1, \dots, P_k such that $\bigcap_i \mathcal{CH}(P_i)$ is not empty. A point in this intersection is a **Tverberg point**. Tverberg's theorem states that P has a Tverberg partition into $\lceil n/(d+1) \rceil$ sets. In particular, the *Tverberg depth* (*T-depth*) of a point q is the maximum size k of a Tverberg partition such that $q \in \bigcap_{i=1}^k \mathcal{CH}(P_i)$.

By definition, points of T-depth $n/(d+1)$ are centerpoints for P . In the plane, Reay [25] showed that if a point has Tukey depth $k \leq |P|/3$, then the T-depth of q is k . This property is already false in three dimensions [4]. The two-dimensional case was handled by Birch [5], who proved that any set of n points in the plane can be partitioned into $n/3$ triples whose induced triangles have a common intersection point.

Computing a Tverberg point

For work on computing approximate Tverberg points, see [23, 24, 26, 9] and the references therein. Currently, no polynomial-time (in both n and d) approximation algorithm is known for computing Tverberg points. This search problem is believed to be quite hard, see [22].

Algorithms for computing an *exact* Tverberg point of T-depth $n/(d+1)$ implement the construction implied by the original proof. The runtime of such an algorithm is $d^{O(d^2)} n^{d(d+1)+1}$, see the full version [17]. As previously mentioned, the exception is in two dimensions, where the algorithm of Birch [5] runs in $O(n \log n)$ time. But even in three dimensions, we are unaware of an algorithm faster than $O(n^{13})$.

Convex combinations and Carathéodory's theorem

The challenge in finding a Tverberg point is that we have few subroutines at our disposal with runtimes polynomial in d . Consider the most basic task – given a set P of n points and a query point q , decide if q lies inside $\mathcal{CH}(P)$, and if so, compute the convex combination of q in term of the points of P . This problem can be reduced to linear programming. Currently, the fastest strongly polynomial LP algorithms run in super-polynomial time $2^{O(\sqrt{d \log d})} + O(d^2 n)$

[10, 21], where d is the number of variables and n is the number of constraints. However, any given convex combination of P representing q can be sparsified in polynomial time into a convex combination using only $d + 1$ points of P . Lemma 22 describes this algorithmic version of Carathéodory's theorem.

Radon partitions in polynomial time

Finding points of T-depth 2 is relatively easy. Any set of $d + 2$ points in \mathbb{R}^d can be partitioned into two disjoint sets whose convex hulls intersect, and a point in the intersection is a *Radon point*. Radon points can be computed in $O(d^3)$ time by solving a linear system with $d + 2$ variables. Almost all the algorithms for finding Tverberg points mentioned above amplify the algorithm for finding Radon points.

Our results

The known and new results are summarized in Table 1.1. In Section 2, we review preliminary information and known results, which include the following.

1. **An exact algorithm.** The proof of Tverberg's theorem is constructive and leads to an algorithm with running time $O(n^{d(d+1)})$. It seems that the algorithm has not been described and analyzed explicitly in the literature. For the sake of completeness, we provide this analysis in the full version [17].
2. **In two dimensions.** Given a set P of n points in the plane and a query point q of Tukey depth k , Birch's theorem [5] implies that q can be covered by $\min(k, \lfloor n/3 \rfloor)$ vertex-disjoint triangles of P . One can compute k and this triangle cover in $O(n + k \log k)$ time, and use them to compute a Tverberg point of depth $\lfloor n/3 \rfloor$ in $O(n \log n)$ time.

In Section 3, we provide improved algorithms for computing Tverberg points and partitions.

1. **Projections in low dimensions.** We use projections to find improved approximation algorithms in dimensions 3 to 7, see Table 1.2. For example, in three dimensions, one can compute a point with T-depth $n/6$ in $O(n \log n)$ time.
2. **An improved quasi-polynomial algorithm.** We modify the algorithm of Miller and Sheehy to use a buffer of free points. Coupled with the algorithm of Mulzer and Werner [24], this idea yields an algorithm that computes a point of T-depth $\geq (1 - \delta)n/2(d + 1)^2$ in $d^{O(\log(d/\delta))}n$ time. This improves the approximation quality of the algorithm of [24] by a factor of $2(d + 1)$, while keeping (essentially) the same running time.
3. **A strongly polynomial algorithm.** In Section 3.3, we present the first strongly polynomial approximation algorithm for Tverberg points, with the following caveats:
 - a. the algorithm is randomized, and might fail,
 - b. one version returns a Tverberg partition, but not a point that lies in its intersection,
 - c. the other (inferior) version returns a Tverberg point and a partition realizing it, but not the convex combination of the Tverberg point for each set in the partition.

Specifically, one can compute a partition of P into $n/O(d^2 \log d)$ sets, such that the intersection of their convex hulls is nonempty (with probability close to one), but without finding a point in the intersection. Alternatively, one can also compute a Tverberg point, but the number of sets in the partition decreases to $n/O(d^3 \log d)$.

■ **Table 1.1** The known and improved results for Tverberg partition. The notation \mathcal{O}_w hides terms with polylogarithmic dependency on size of the numbers, see Remark 21. The parameter δ can be freely chosen.

Depth	Running time	Ref / Comment
$n/(d+1)$	$d^{O(d^2)} n^{d(d+1)+1}$	Tverberg theorem
$d = 2 : n/3$	$O(n \log n)$	[5]: Theorem 8
$\frac{n}{2(d+1)^2}$	$n^{O(\log d)}$	Miller and Sheehy [23]
$\frac{n}{4(d+1)^3}$	$d^{O(\log d)} n$	Mulzer and Werner [24]
$\frac{n}{2d(d+1)^2}$	$\mathcal{O}_w(n^4)$	Rolnick and Soberón [26]
$\frac{(1-\delta)n}{d(d+1)}$	$(d/\delta)^{O(d)} + \mathcal{O}_w(n^4)$	Rolnick and Soberón [26]

New results

$\frac{(1-\delta)n}{2(d+1)^2}$	$d^{O(\log(d/\delta))} n$	Theorem 18
$\frac{n}{O(d^2 \log d)}$	$O(dn)$	Lemma 19: Only partition
$\frac{n}{O(d^3 \log d)}$	$O(dn + d^7 \log^6 d)$	Lemma 20: Partition + point, but no convex combination
$\frac{n}{O(d^2 \log d)}$	$\mathcal{O}_w(n^{5/2} + nd^3)$	Lemma 24: Weakly polynomial
$\frac{(1-\delta)n}{2(d+1)^2}$	$d^{O(\log \log(d/\delta))} \mathcal{O}_w(n^{5/2})$	Theorem 25: Weakly quasi polynomial
$\frac{(1-\delta)n}{d(d+1)}$	$O(c + c'n + d^2 n \log^2 n)$ $c = d^{O(d)} / \delta^{2(d-1)}$ $c' = 2^{O(\sqrt{d \log d})}$	Lemma 26: Useful for low dimensions

■ **Table 1.2** The best approximation ratios for Tverberg depth in low dimensions, with nearly linear-time algorithms, as implied by Lemma 16. Note that the new algorithm is no longer an improvement in dimension 8. We are unaware of any better approximation algorithms (except for running the exact algorithm for Tverberg's point, which requires $n^{O(d^2)}$ time).

Dim	T. Depth	New depth	Known	Ref	Comment
3	$n/4$	$n/6$	$n/8$	[24]	
4	$n/5$	$n/9$	$n/16$		
5	$n/6$	$n/18$	$n/32$		
6	$n/7$	$n/27$	$(1-\delta)n/42$	[26]	Original paper describes a weakly polynomial algorithm. The improved algorithm is described in Lemma 26.
7	$n/8$	$n/54$	$(1-\delta)n/56$		
8	$n/9$	$n/81$	$(1-\delta)n/72$		

4. **A weakly polynomial algorithm.** Revisiting an idea of Rolnick and Soberón [26], we use algorithms for solving LPs. The resulting running time is either weakly polynomial (depending logarithmically on the relative sizes of the numbers in the input) or super-polynomial, depending on the LP solver. In particular, the randomized, strongly polynomial algorithms described above can be converted into constructive algorithms that compute the convex combination of the Tverberg point over each set in its partition. Having computed approximate Tverberg points of T-depth $\geq n/O(d^2 \log d)$, we can feed them into the buffered version of Miller and Sheehy's algorithm to compute Tverberg points of depth $\geq (1 - \delta)n/2(d + 1)^2$. This takes $d^{O(\log \log(d/\delta))} \mathcal{O}_w(n^{5/2})$ time, where \mathcal{O}_w hides polylogarithmic terms in the size of the numbers involved, see Remark 21.
5. **Faster approximation in low dimensions.** One can compute (or approximate) a centerpoint, then repeatedly extract simplices covering it until the centerpoint is exposed. This leads to an $O_d(n^2)$ approximation algorithm [26]. Since O_d hides constants that depend badly on d , this method is most useful in low dimensions. By random sampling, we can speed up this algorithm to $O_d(n \log^2 n)$ time.

2 Background, preliminaries and known results

► **Definition 1.** A *Tverberg partition* (or a *log*) of a set $P \subseteq \mathbb{R}^d$, for a point q , is a set $\ell = \{P_1, \dots, P_k\}$ of vertex-disjoint subsets of P , each containing at most $d + 1$ points, such that $q \in \mathcal{CH}(P_i)$, for all i . The **rank** of ℓ is $k = |\ell|$. The maximum rank of any log of q is its **Tverberg depth** (**T-depth**).

A set P_j in a log is a **batch**. For every batch $P_j = \{p_1, \dots, p_{d+1}\}$ in the log, we store the convex coefficients $\alpha_1, \dots, \alpha_{d+1} \geq 0$ such that $\sum_i \alpha_i p_i = q$ and $\sum_i \alpha_i = 1$. A pair (q, ℓ) of a point and its log is a **site**.

Tverberg's theorem states that, for any set of n points in \mathbb{R}^d , there is a point in \mathbb{R}^d with Tverberg depth $\lceil n/(d + 1) \rceil$. For simplicity, we assume the input is in general position.

2.1 An exact algorithm

The constructive proof of Tverberg and Vrećica [28] implies an algorithm for computing exact Tverberg points – see [17] for details.

► **Lemma 2.** Let P be a set of n points in \mathbb{R}^d . In $d^{O(d^2)} n^{d(d+1)+1}$ time, one can compute a point q and a partition of P into disjoint sets P_1, \dots, P_r such that $q \in \bigcap_i \mathcal{CH}(P_i)$, where $r = \lceil n/(d + 1) \rceil$.

2.2 In two dimensions

We first review Tukey depth, which is closely related to Tverberg depth in two dimensions.

► **Definition 3.** The **Tukey depth** of a point q in a set $P \subseteq \mathbb{R}^d$, denoted by $d_{\text{TK}}(q)$, is the minimum number of points contained in any closed halfspace containing q .

The following result is implicit in the proof of Theorem 5.2 in [6]. Chan solves the decision version of the Tukey depth problem, while we need to compute it explicitly, resulting in a more involved algorithm. For the sake of completeness, we provide the details in Appendix A.

► **Lemma 4** (Proof in Appendix A). Given a set P of n points in the plane and a query point q , such that $P \cup \{q\}$ is in general position, one can compute, in $O(n + k \log k)$ time, the Tukey depth k of q in P . The algorithm also computes the halfplane realizing this depth.

For shallow points in the plane, the Tukey and Tverberg depths are equivalent, and we can compute the associated Tverberg partition.

► **Lemma 5** (Proof in Appendix B). *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = d_{TK}(q) \leq n/3$. Then one can compute a log for q of rank k in $O(n + k \log k)$ time.*

The Tukey depth of a point can be as large as $\lfloor n/2 \rfloor$. Indeed, consider the vertices of a regular n -gon (for odd n), the polygon center has depth $\lfloor n/2 \rfloor$.

► **Lemma 6**. *Let P be a set of n points in the plane, and let q be a query point such that $P \cup \{q\}$ is in general position, and q has Tukey depth larger than $n/3$. Then, one can compute a log for q of this rank in $O(n \log n)$ time.*

Proof. Let $N = \lfloor n/3 \rfloor$. Assume q that is the origin, and sort the points of P in counter-clockwise order, where p_i is the i th point in this order, for $i = 1, \dots, n$. For $i = 1, \dots, N$, let $\Delta_i = \Delta_{p_i p_{N+i} p_{2N+i}}$. We claim that $\angle p_i q p_{N+i} \leq \pi$. Otherwise, there is a halfspace containing fewer than N points induced by the line passing through p_i and q . Similarly, $\angle p_{N+i} q p_{2N+i} \leq \pi$ and $\angle p_{2N+i} q p_i \leq \pi$, so that q lies inside Δ_i , as desired. ◀

► **Theorem 7**. *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = d_{TK}(q)$ is the Tukey depth of q in P . Then one can compute the Tukey depth k of q , along with a log of rank $\tau = \min(\lfloor n/3 \rfloor, k)$, in $O(n + k \log k)$ time.*

Proof. Compute the Tukey depth of q in $O(n + k \log k)$ time, using the algorithm of Lemma 4. If $k \leq n/3$, then compute the log using the algorithm of Lemma 5, and otherwise using the algorithm of Lemma 6. ◀

The above implies the following theorem of Birch, which predates Tverberg's theorem.

► **Theorem 8** ([5]). *Let P be a set of $n = 3k$ points in the plane. Then there exists a partition of P into k vertex-disjoint triangles, such that their intersection is not empty. The partition can be computed in $O(n \log n)$ time.*

Proof. A centerpoint of P can be computed in $O(n \log n)$ time [7]. Such a centerpoint has Tukey depth at least k , so the algorithm of Lemma 6 partitions P into k triangles. ◀

► **Remark 9.** (A) Note that Tverberg's theorem in the plane is slightly stronger – it states that any point set with $3k - 2$ points has Tverberg depth k . In such a decomposition, some of the sets may be pairs of points or singletons.

(B) In the colored version of Tverberg's theorem in the plane, one is given $3n$ points partitioned into three classes of equal size. Agarwal *et al.* [3] showed how to compute a decomposition into n triangles covering a query point, where every triangle contains a vertex of each color (if such a decomposition exists). This problem is significantly more difficult, and their running time is a prohibitive $O(n^{11})$.

2.3 Miller and Sheehy's algorithm

Here, we review Miller and Sheehy's [23] approximation algorithm for computing a Tverberg point before describing our improvement.

Radon partitions

Radon's theorem states that a set P of $d+2$ points in \mathbb{R}^d can be partitioned into two disjoint subsets P_1, P_2 such that $\mathcal{CH}(P_1) \cap \mathcal{CH}(P_2) \neq \emptyset$. This partition can be computed via solving a linear system in $d+2$ variables in $O(d^3)$ time. A point in this intersection (which is an immediate byproduct of computing the partition) is a **Radon point**.

Let p be a point in \mathbb{R}^d . It is a **convex combination** of points $\{p_1, \dots, p_m\}$ if there are $\alpha_1, \dots, \alpha_m \in [0, 1]$ such that $p = \sum_{i=1}^m \alpha_i p_i$ and $\sum_i \alpha_i = 1$.

► **Lemma 10** ([23, 17]: Sparsifying conv. combination). *Let p be a point in \mathbb{R}^d , and let $P = \{p_1, \dots, p_m\} \subseteq \mathbb{R}^d$ be a point set. Furthermore, assume that we are given p as a convex combination of points of P . Then, one can compute a convex combination representation of p that uses at most $d+1$ points of P . This takes $O(md^3)$ time.*

► **Lemma 11** ([23, 17]). *Given $d+2$ sites $(p_1, \ell_1), \dots, (p_{d+2}, \ell_{d+2})$ of rank r in \mathbb{R}^d , where the logs ℓ_i are disjoint, one can compute a site (p, ℓ) of rank $2r$ in $O(rd^5)$ time.*

A recycling algorithm for computing a Tverberg point

The algorithm of Miller and Sheehy maintains a collection of sites. Initially, it converts each input point $p \in P$ into a site $(p, \{p\})$ of rank one. The algorithm then merges $d+2$ sites of rank r into a site of rank $2r$, using Lemma 11. Before the merge, the input logs use $r(d+2)(d+1)$ points in total. After the merge, the new log uses only $2(d+1)r$ points. The algorithm recycles the remaining $(d+2)(d+1)r - 2(d+1)r = d(d+1)r$ points by reinserting them into the collection as singleton sites of rank one. When no merges are available, the algorithm outputs the maximum-rank site as the approximate Tverberg point.

Analysis

For our purposes, we need a slightly different way of analyzing the above algorithm of Miller and Sheehy [23], so we provide the analysis in detail.

Let 2^h be the rank of the output site. The number of points in all the logs is n . Since a site of rank 2^i has at most $(d+1)2^i$ points of P in its log, and there are at most $d+1$ sites of each rank, n is at most $\sum_{i=0}^h (d+1)2^{2^i}$. As such,

$$n \leq \sum_{i=0}^h (d+1)2^{2^i} = (d+1)^2(2^{h+1} - 1) \iff \frac{n}{(d+1)^2} + 1 \leq 2^{h+1} \implies 2^h \geq \left\lceil \frac{n}{2(d+1)^2} \right\rceil. \quad (2.1)$$

Every site in the collection is associated with a history tree that describes how it was computed. Thus, the algorithm execution generates (conceptually) a forest of such history trees, which is the **history** of the computation.

► **Lemma 12.** *Let 2^h be the maximum rank of a site computed by the algorithm. The total number of sites in the history that are of rank (exactly) 2^{h-i} is at most $(d+2)^{i+1} - 1$.*

Proof. There are at most $T(0) = d+1$ nodes of rank 2^{h-0} in the history, and each has $d+2$ children of rank 2^{h-1} . In addition, there might be $d+1$ trees in the history whose roots have rank 2^{h-1} . Thus, $T(1) = d+1 + (d+2)T(0)$. By induction, there are at most

$$T(i) = d+1 + (d+2)T(i-1) = (d+2)^{i+1} - 1$$

nodes in the history of rank 2^{h-i} . ◀

► **Lemma 13.** *Let 2^h be the maximum rank of a site computed by the algorithm. The total amount of work spent (directly) by the algorithm (throughout its execution) at nodes of rank $\geq 2^{h-i}$ is $O(d^{4+i}n)$. In particular, the overall running time of the algorithm is $O(d^4n^{1+\log_2 d})$.*

Proof. There are at most $(d+2)^{i+1}$ nodes of rank 2^{h-i} in the history. The rank of such a node is $r_i = O(\frac{n}{2^{i+1}(d+1)^2})$, and the amount of work spent in computing the node is $O(r_i d^6) = O(d^4n/2^i)$. As such, the total work in the top i ranks of the history is proportional to $L_i = \sum_{j=0}^i (d+2)^j \cdot d^4n/2^j = O(d^{4+i}n)$.

Since $h \leq \log_2 n$, the overall running time is $O(L_h) = O(d^{4+\log_2 n}n) = O(d^4n^{1+\log_2 d})$. ◀

3 Improved Tverberg approximation algorithms

3.1 Projections in low dimensions

► **Lemma 14.** *Let P be a set of n points in three dimensions. One can compute a Tverberg point of P of depth $n/6$ (and the log realizing it) in $O(n \log n)$ time.*

Proof. Project the points of P to two dimensions, and compute a Tverberg point p and a partition for it of size $n/3$. Lifting from the plane back to the original space, the point p lifts to a vertical line ℓ , and every triangle lifts to a triangle which intersects ℓ . Pick a point q on this line which is the median of the intersections. Now, pair every triangle intersecting ℓ above q with a triangle intersecting ℓ below it. This partitions P into $n/6$ sets, each of size 6, such that the convex hull of each sets contains q . Within each set, compute at most 4 points whose convex hull contains q . These points yield the desired log for q of rank $n/6$. ◀

► **Remark 15.** Lemma 14 seems innocent enough, but to the best of our knowledge, it is the best one can do in near-linear time in 3D. The only better approximation algorithm we are aware of is the one suggested by Tverberg's theorem. It yields a point of Tverberg depth $n/4$, but its running time is probably at least $O(n^{13})$ (see Lemma 2).

As observed by Mulzer and Werner [24], one can repeat this projection mechanism. Since Mulzer and Werner [24] bottom their recursion at dimension 1, their algorithm computes a point of Tverberg depth $n/2^d$ (in three dimensions, depth $n/8$). Applying this projection idea but bottoming at two dimensions, as above, yields a point of Tverberg depth $n/(3 \cdot 2^{d-2})$.

► **Lemma 16.** *Let P be a set of n points in four dimensions. One can compute a Tverberg point of P of depth $n/9$ (and the log realizing it) in $O(n \log n)$ time.*

More generally, for d even, and a set P of n points in \mathbb{R}^d , one can compute a point of depth $n/3^{d/2}$ in $d^{O(1)}n \log n$ time. For d odd, we get a point of depth $n/(2 \cdot 3^{(d-1)/2})$.

Proof. As mentioned above, the basic idea is due to Mulzer and Werner. Project the four-dimensional point set onto the plane spanned by the first two coordinates (i.e. “eliminate” the last two coordinates), and compute a $n/3$ centerpoint using Theorem 8. Translate the space so that this centerpoint lies at the origin. Now, consider each triangle in the original four-dimensional space. Each triangle intersects the two-dimensional subspace formed by the first two coordinates. Pick an intersection point from each lifted triangle. On this set of $n/3$ points, living in this two dimensional subspace, apply again the algorithm of Theorem 8 to compute a Tverberg point of depth $(n/3)/3 = n/9$. The resulting centerpoint p is now contained in $n/9$ triangles, where every vertex is contained in an original triangle of points. That is, p has depth $n/9$, where each group consists of 9 points in four dimensions. Now, sparsify each group into 5 points whose convex hull contains p .

The second part of the claim follows from applying the above argument repeatedly. ◀

3.2 An improved quasi-polynomial algorithm

The algorithm of Miller and Sheehy is expensive at the bottom of the recursion tree, so we replace the bottom with a faster algorithm. We use a result of Mulzer and Werner [24]:

► **Theorem 17** ([24]). *Given a set P of n points in \mathbb{R}^d , one can compute a site of rank $\geq n/4(d+1)^3$ (together with its log) in $d^{O(\log d)}n$ time.*

Let $\delta \in (0, 1)$ be a small constant. We modify the algorithm of Miller and Sheehy by keeping all the singleton sites of rank one in a buffer of *free* points. Initially, all points are in the buffer. Whenever the buffer contains at least δn points, we use the algorithm of Theorem 17 to compute a site of rank $\rho \geq \delta n/8(d+1)^3$, where ρ is a power of two. (If the computed rank is too large, we throw away entries from the log until the rank reaches a power of two.) We insert this site into the collection of sites maintained by the algorithm. As in Miller and Sheehy's algorithm, we repeatedly merge $d+2$ sites of the same rank to get a site of double the rank. In the process, the points thrown out from the log are recycled into the buffer. Whenever the buffer size exceeds δn , we compute a new site of rank ρ . This process stops when no sites can be merged and the number of free points is less than δn .

► **Theorem 18.** *Given a set P of n points in \mathbb{R}^d , and a parameter $\delta \in (0, 1)$, one can compute a site of rank at least $\frac{(1-\delta)n}{2(d+1)^2}$ (together with its log) in $d^{O(\log(d/\delta))}n$ time.*

Proof. When the algorithm above stops, there are at least $(1-\delta)n$ points in its logs. Arguing as in Eq. (2.1), the output site has rank

$$2^h \geq \frac{(1-\delta)n}{2(d+1)^2}.$$

We now consider runtime. The algorithm maintains only nodes of rank $\rho \geq 2^{h-H}$, where

$$H = \left\lceil \log_2 \frac{2^h}{\rho} \right\rceil \leq \left\lceil \log_2 \frac{n/2(d+1)^2}{\delta n/8(d+1)^3} \right\rceil = 1 + \log_2 \frac{4(d+1)}{\delta} = O(\log(d/\delta)).$$

The total work spent on merging nodes with these ranks is equivalent to the work in Miller and Sheehy's algorithm for such nodes. By Lemma 13, the total work performed is $O(d^{4+H}n)$.

As for the work associated with the buffer, observe that Theorem 17 is invoked $(d+2)^{H+1}$ times (this is the number of nodes in the history of rank 2^{h-H}). Each invocation takes $d^{O(\log d)}n$ time, so the total running time of the algorithm is

$$d^{O(H)}n + (d+1)^{H+1}d^{O(\log d)}n = d^{O(\log(d/\delta))}n. \quad \blacktriangleleft$$

3.3 A strongly polynomial algorithm

► **Lemma 19.** *Let P be a set of n points in \mathbb{R}^d . For $N = \Theta(d^2 \log d)$, consider a random coloring of P by $k = \lfloor n/N \rfloor$ colors, and let $\{P_1, \dots, P_k\}$ be the resulting partition of P . With probability $\geq 1 - k/d^{O(d)}$, this partition is a Tverberg partition of P .*

Proof. Let $\varepsilon = 1/(d+1)$. The VC dimension of halfspaces in \mathbb{R}^d is $d+1$ by Radon's theorem. By the ε -net theorem [18, 15], a sample from P of size

$$N = \left\lceil \frac{8(d+1)}{\varepsilon} \log(16(d+1)) \right\rceil = \Theta(d^2 \log d)$$

is an ε -net with probability $\geq 1 - 4(16(d+1))^{-2(d+1)}$. Then P_1, \dots, P_k are all ε -nets with the probability stated in the lemma.

Now, consider the centerpoint q of P . We claim that $q \in \mathcal{CH}(P_i)$, for all i . Indeed, assume otherwise, so that there is a separating hyperplane between q and some P_i . Then the halfspace induced by this hyperplane that contains q also contains at least εn points of P , because q is a centerpoint of P . But this contradicts that P_i is an ε -net for halfspaces. ◀

► **Lemma 20.** *Let P be a set of n points in \mathbb{R}^d . For $N = O(d^3 \log d)$, consider a random coloring of P by $k = \lfloor n/N \rfloor = \Omega(\frac{n}{d^3 \log d})$ colors, and let $\{P_1, \dots, P_k\}$ be the resulting partition of P . One can compute, in $O(d^7 \log^6 d)$ time, a Tverberg point that lies in $\bigcap_i \mathcal{CH}(P_i)$. This algorithm succeeds with probability $\geq 1 - k/d^{O(d)}$.*

Proof. Compute a $1/(2d^2)$ -centerpoint q for P using the algorithm of Har-Peled and Jones [16], and repeat the argument of Lemma 19 with $\varepsilon = 1/(2d^2)$. With the desired probability, $q \in \mathcal{CH}(P_i)$ for all i . ◀

3.4 A weakly polynomial algorithm

Let $T_{LP}(n, m)$ be the time to solve an LP with n variables and m constraints. Using interior point methods, one can solve such LPs in weakly polynomial time. The fastest known method runs in $\mathcal{O}_w(mn + n^3)$ [29] or $\mathcal{O}_w(mn^{3/2})$ [20], where \mathcal{O}_w hides polylogarithmic terms that depends on n, m , and the width of the input numbers.

► **Remark 21.** The *error* of the LP solver, see [29], for a prescribed parameter ε , is the distance of the computed solution from an optimal one. Specifically, let R be the maximum absolute value of any number in the given instance. In $O((nm + n^3) \log^{O(1)} n \log(n/\varepsilon))$ time, the LP solver can find an assignment to the variables which is \mathcal{E} close to complying with the LP constraints, where $\mathcal{E} \leq \varepsilon n m R$ [29]. That is, the LP solver can get arbitrarily close to a true solution. This is sufficient to compute an exact solution in polynomial time if the input is made out of integer numbers with polynomially bounded values, as the running time then depends on the number of bits used to encode the input. We use $\mathcal{O}_w(\cdot)$ to denote such **weakly polynomial** running time.

► **Lemma 22** (Carathéodory via LP). *Given a set P of n points in \mathbb{R}^d and query point q , one can decide if $q \in \mathcal{CH}(P)$, and if so, output a convex combination of $d + 1$ points of P that is equal to q . The running time of this algorithm is $O(T_{LP}(n, n + d) + nd^3) = \mathcal{O}_w(nd + nd^3)$.*

Alternatively, one can compute such a point in $2^{O(\sqrt{d \log d})} + O(d^2 n)$ time.

Proof. We write the natural LP for representing q as a point in the interior of $\mathcal{CH}(P)$. This LP has n variables (one for each point), and $n + 2d + 2$ constraints (all variables are positive, the points sum to q , and the coefficients sum to 1), where an equality constraint counts as two constraints. If the LP computes a solution, then we can sparsify it using Lemma 10.

The alternative algorithm writes an LP to find a hyperplane separating q from P . First, it tries to find a hyperplane which is vertically below P and above q . This LP has d variables, and it can be solved in $O(2^{\sqrt{d \log d}} + d^2 n)$ time [27, 10, 21]. If there is no such separating hyperplane, then the algorithm provides d points whose convex hull lies below q and tries again. This time, it computes a separating hyperplane below q and above P . Again, if no such separating hyperplane exists, then the algorithm returns d points whose convex hull lies above q . The union of the computed point sets above and below q contains at most $2d$ points. Now, write the natural LP with $2d$ variables as above, and solve it using linear-time LP solvers in low dimensions. This gives the desired representation. ◀

► **Remark 23.** Rolnick and Soberón [26] achieved a result similar to that of Lemma 22, but they used binary search to find the $d + 1$ coefficients in the minimal representation. Thus their running time is $O(T_{LP}(n, n + d)d \log n)$.

Using Lemma 19, we get the following.

► **Lemma 24.** *Let P be a set of n points in \mathbb{R}^d . For $N = O(d^2 \log d)$, one can compute a site q of rank $k = n/O(d^2 \log d)$ in $\mathcal{O}_w(n^{5/2} + nd^3)$ time. The algorithm succeeds with probability $\geq 1 - k/d^{O(d)}$.*

Proof. We compute a Tverberg partition using Lemma 19. Next we write an LP for computing an intersection point q that lies in the interior of the $k = n/O(d^2 \log d)$ sets. This LP has $O(n + d)$ constraints and n variables, and it can be solved in $\mathcal{O}_w(n^{5/2})$ time [20]. We then sparsify the representations over each of the k sets. This requires $O(d^2 \log d \cdot d^3)$ time per set and hence $O(nd^3)$ time overall. The result is a site q with a log of rank k , as desired. ◀

► **Theorem 25.** *Given a set P of n points in \mathbb{R}^d , and a parameter $\delta \in (0, 1)$, one can compute a site of rank at least $\frac{(1-\delta)n}{2(d+1)^2}$ (together with its log) in $d^{O(\log \log(d/\delta))} \mathcal{O}_w(n^{5/2})$ time.*

Proof. We modify the algorithm of Theorem 18 to use Lemma 24 to compute a Tverberg point on the points in the buffer. Since the gap between the top rank of the recursion tree and rank computed by Lemma 24 is $O(\log(d/\delta))$, it follows that the algorithm uses only the top $O(\log \log(d/\delta))$ levels of the recursion tree, and the result follows. ◀

3.5 Faster approximation in low dimensions

► **Lemma 26.** *Let P be a set of n points in \mathbb{R}^d , and let $\delta \in (0, 1)$ be some parameter. One can compute a Tverberg point of depth $\geq (1 - \delta)n/d(d + 1)$, together with its log, in $O(d^{O(d)}/\delta^{2(d-1)} + 2^{O(\sqrt{d \log d})}n + d^2 n \log^2 n)$ time. The algorithm succeeds with probability close to one.*

Proof. Let $\varepsilon = 1/2(d + 1)$, and let R be a random sample from P of size

$$m = O(d\varepsilon^{-1}\delta^{-2} \log \varepsilon^{-1}) = O(d^2\delta^{-2} \log d).$$

This sample is a (ε, δ) -relative approximation for P with probability close to one [14]. Compute a centerpoint c for the sample using “brute force” in $O(m^{d-1})$ time [7]. Now, repeatedly use the algorithmic version of Carathéodory’s theorem (Lemma 22) to extract a simplex that contains c . One can repeat this process $\lfloor n/d(d + 1) \rfloor$ times before getting stuck, since every simplex contains at most d points in any halfspace passing through c . Naively, the running time of this algorithm is $2^{O(\sqrt{d \log d})}n^2$.

However, one can do better. The number of simplices extracted by the above algorithm is $L = \lfloor n/d(d + 1) \rfloor$. For $i = 1, \dots, L$, let $b_i = \lfloor n/(d + 1) \rfloor - d(i - 1)$ be a lower bound on the Tukey depth of c at the beginning of the i th iteration of the above extraction algorithm. At this moment, the point set has $n_i = n - (d + 1)(i - 1) \geq n/2$ points. Hence the relative Tukey depth of c is $\varepsilon_{L-i} = b_{L-i}/n_{L-i}$. In particular, an ε_i -net R_i for halfspaces has size $r_i = O(\frac{d}{\varepsilon_i} \log \frac{1}{\varepsilon_i})$. If r_i is larger than the number of remaining points, then the sample consists of the remaining points of P . The convex hull of such a sample contains c with probability close to one, so one can apply Lemma 22 to R_i to get a simplex that contains P (if the sample fails, then the algorithm resamples). The algorithm adds the simplex to the output log, removes its vertices from P , and repeats.

Since the algorithm invokes Lemma 22 $O(n/d^2)$ times, the running time is bounded by $2^{O(\sqrt{d \log d})} n/d^2 + O(d^2 \sum_i r_i)$. Since $n_{L-i} \geq n/2$ and $b_{L-i} \geq i \cdot d$, we have that $\varepsilon_{L-i} \geq (id/(n/2)) = 2id/n$. Therefore,

$$\sum_i r_i = O(n) + \sum_{i=1}^{L-1} r_{L-i} = O(n) + \sum_{i=1}^{L-1} O\left(\frac{dn}{2id} \log \frac{n}{id}\right) = O(n \log^2 n). \quad \blacktriangleleft$$

References

- 1 Peyman Afshani, Donald R. Sheehy, and Yannik Stein. Approximating the simplicial depth. *CoRR*, abs/1512.04856, 2015. [arXiv:1512.04856](#).
- 2 Pankaj K. Agarwal, Sarel Har-Peled, Subhash Suri, Hakan Yildiz, and Wuzhou Zhang. Convex hulls under uncertainty. *Algorithmica*, 79(2):340–367, 2017. doi:10.1007/s00453-016-0195-y.
- 3 Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. Algorithms for center and Tverberg points. *ACM Trans. Algo.*, 5(1):5:1–5:20, 2008. doi:10.1145/1435375.1435380.
- 4 David Avis. The m -core properly contains the m -divisible points in space. *Pattern Recognit. Lett.*, 14(9):703–705, 1993. doi:10.1016/0167-8655(93)90138-4.
- 5 B. J. Birch. On 3N points in a plane. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):289–293, 1959. doi:10.1017/S0305004100034071.
- 6 T. M. Chan. Geometric applications of a randomized optimization technique. *Disc. Comput. Geom.*, 22(4):547–567, 1999. doi:10.1007/PL00009478.
- 7 T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In J. Ian Munro, editor, *Proc. 15th ACM-SIAM Sympos. Discrete Algs. (SODA)*, pages 430–436. SIAM, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982853>.
- 8 B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31(4):509–517, 1985. doi:10.1109/TIT.1985.1057060.
- 9 Aruni Choudhary and Wolfgang Mulzer. No-dimensional Tverberg theorems and algorithms. In Sergio Cabello and Danny Z. Chen, editors, *Proc. 36th Int. Annu. Sympos. Comput. Geom. (SoCG)*, volume 164 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.31.
- 10 K. L. Clarkson. Las Vegas algorithms for linear and integer programming. *J. Assoc. Comput. Mach.*, 42:488–499, 1995. doi:10.1145/201019.201036.
- 11 Kenneth L. Clarkson, David Eppstein, Gary L. Miller, Carl Sturtivant, and Shang-Hua Teng. Approximating center points with iterative Radon points. *Int. J. Comput. Geom. Appl.*, 6:357–377, 1996. doi:10.1142/S021819599600023X.
- 12 Ketan Dalal. Counting the onion. *Random Struct. Alg.*, 24(2):155–165, 2004. doi:10.1002/rsa.10114.
- 13 S. Har-Peled and B. Lidicky. Peeling the grid. *SIAM J. Discrete Math.*, 27(2):650–655, 2013. doi:10.1137/120892660.
- 14 S. Har-Peled and M. Sharir. Relative (p, ε) -approximations in geometry. *Disc. Comput. Geom.*, 45(3):462–496, 2011. doi:10.1007/s00454-010-9248-1.
- 15 Sarel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.
- 16 Sarel Har-Peled and Mitchell Jones. Journey to the center of the point set. In Gill Barequet and Yusu Wang, editors, *Proc. 35th Int. Annu. Sympos. Comput. Geom. (SoCG)*, volume 129 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.41.
- 17 Sarel Har-Peled and Timothy Zhou. Improved approximation algorithms for Tverberg partitions. *CoRR*, abs/2007.08717, 2020. [arXiv:2007.08717](#).
- 18 D. Haussler and E. Welzl. ε -nets and simplex range queries. *Disc. Comput. Geom.*, 2:127–151, 1987. doi:10.1007/BF02187876.

- 19 Shreesh Jadhav and Asish Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Disc. Comput. Geom.*, 12:291–312, 1994. doi:10.1007/BF02574382.
- 20 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In Venkatesan Guruswami, editor, *Proc. 56th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 230–249. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.23.
- 21 J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996. doi:10.1007/BF01940877.
- 22 Frédéric Meunier, Wolfgang Mulzer, Pauline Sarrazolles, and Yannik Stein. The rainbow at the end of the line - A PPAD formulation of the colorful carathéodory theorem with applications. In Philip N. Klein, editor, *Proc. 28th ACM-SIAM Sympos. Discrete Algs. (SODA)*, pages 1342–1351. SIAM, 2017. doi:10.1137/1.9781611974782.87.
- 23 Gary L. Miller and Donald R. Sheehy. Approximate centerpoints with proofs. *Comput. Geom.*, 43(8):647–654, 2010. doi:10.1016/j.comgeo.2010.04.006.
- 24 Wolfgang Mulzer and Daniel Werner. Approximating Tverberg points in linear time for any fixed dimension. *Disc. Comput. Geom.*, 50(2):520–535, 2013. doi:10.1007/s00454-013-9528-7.
- 25 John R. Reay. Several generalizations of Tverberg’s theorem. *Israel Journal of Mathematics*, 34(3):238–244, 1979. doi:10.1007/BF02760885.
- 26 David Rolnick and Pablo Soberón. Algorithms for Tverberg’s theorem via centerpoint theorems. *CoRR*, abs/1601.03083, 2016. arXiv:1601.03083.
- 27 R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Disc. Comput. Geom.*, 6:423–434, 1991. doi:10.1007/BF02574699.
- 28 Helge Tverberg and Siniša Vrećica. On generalizations of Radon’s theorem and the ham sandwich theorem. *Euro. J. Combin.*, 14(3):259–264, 1993. doi:10.1006/eujc.1993.1029.
- 29 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proc. 52nd Annu. ACM Sympos. Theory Comput. (STOC)*, pages 775–788. ACM, 2020. doi:10.1145/3357713.3384309.

A Tukey depth in two dimensions

Restatement of Lemma 4. *Given a set P of n points in the plane and a query point q , such that $P \cup \{q\}$ is in general position, one can compute, in $O(n + k \log k)$ time, the Tukey depth k of q in P . The algorithm also computes the halfplane realizing this depth.*

Proof. In the dual, q^* is a line, and the task is to find a point on this line which minimizes the number of lines of P^* (i.e., set of lines dual to the points of P) strictly below it. More precisely, one has to also solve the upward version, and return the minimum of the two solutions. Handling the downward version first, every point $p \in P$ has a dual line p^* . The portion of q^* that lies above p^* is a closed ray on q^* . As such, we have a set of rays on the line (which can be interpreted as the x -axis), and the task is to find a point on the line contained in the minimum number of rays. (This is known as linear programming with violations in one dimension.)

Let R_{\Rightarrow} (resp. L_{\Leftarrow}) be the set of points that corresponds to heads of rays pointing to the right (resp. to the left) by P^* on q^* . Let R_i be the set of $\lfloor n/2^i \rfloor$ rightmost points of R_{\Rightarrow} , for $i = 0, \dots, h = \lceil \log_2 n \rceil$. Using median selection, each set R_i can be computed from R_{i-1} in $|R_{i-1}|$ time. As such, all these sets can be computed in $\sum_i O(n/2^i) = O(n)$ time. The sets L_0, L_1, \dots, L_h are computed in a similar fashion. For all i , we also compute the rightmost point r_{i-1} of $R_{i-1} \setminus R_i$ (which is the rightmost point in $R \setminus R_i$). Similarly, ℓ_i is the leftmost point of $L_{i-1} \setminus L_i$, for $i = 1, \dots, h$. Let r_0 (resp. ℓ_0) be the rightmost (resp. leftmost) point of R_0 (resp. L_0).

Now, compute the maximum j such that r_j is to the left of ℓ_j . Observe that $(R \setminus R_j) \cup (L \setminus L_j)$ form a set of rays that their intersection is non-empty (i.e., feasible). Similarly, the set of rays $(R \setminus R_{j+1}) \cup (L \setminus L_{j+1})$ is not feasible, and any set of feasible rays must be created by removing at least $|R_{j+1}| = |L_{j+1}| = \lfloor n/2^{j+1} \rfloor$ rays from $R_{\Rightarrow} \cup L_{\Leftarrow}$. Hence, in linear time, we have computed a 4-approximation to the minimum number of rays that must be removed for feasibility. In particular, if S is a set of rays such that $(R_{\Rightarrow} \cup L_{\Leftarrow}) \setminus S$ is feasible, then $(R_{j-1} \cup L_{j-1}) \setminus S$ is also feasible. Namely, if the minimum size of such a removeable set S is k , then we have computed a set of $O(k)$ rays, such that it suffices to solve the problem on this smaller set.

In the second stage, we solve the problem on $R_{j-1} \cup L_{j-1}$. We first sort the points, and then for each point in this set, we compute how many rays must be removed before it lies in the intersection of the remaining rays. Given a location p on the line, we need to remove all the rays of R_{j-1} (resp. L_{j-1}) whose heads lie to the right (resp. left) of p . This can be done in $O(k \log k)$ time by sweeping from left to right and keeping track of the rays that need to be removed.

As such, we can solve the LP with violations on the line in $O(n + k \log k)$ time, where k is the minimum number of violated constraints. A 4-approximation to k can be computed in $O(n)$ time.

Now we return to the Tukey depth problem. First we compute a 4-approximation, denoted by \tilde{k}_{\downarrow} , for the minimum number of lines crossed by a vertical ray shot down from a point on q^* . Similarly, we compute \tilde{k}_{\uparrow} . If $4\tilde{k}_{\downarrow} < \tilde{k}_{\uparrow}$, then we compute k_{\downarrow} exactly and return the point on q^* that realizes it. (In the primal, this corresponds to a closed halfspace containing q along with exactly k_{\downarrow} points of P .) Similarly, if $\tilde{k}_{\downarrow} > 4\tilde{k}_{\uparrow}$, then we compute k_{\uparrow} exactly, and return it as the desired solution. In the remaining case, we compute both quantities and return the minimum of the two. ◀

B Tverberg partitions in two dimensions

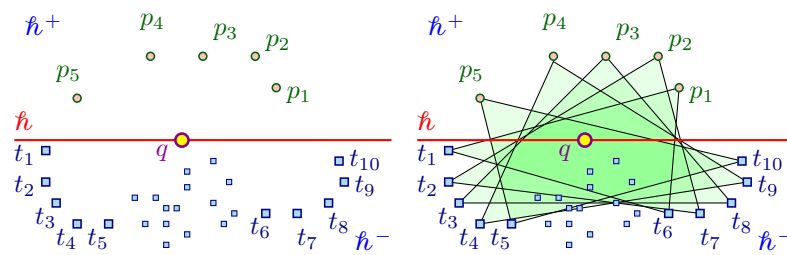
Restatement of Lemma 5. *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = d_{TK}(q) \leq n/3$. Then one can compute a log for q of rank k in $O(n + k \log k)$ time.*

Proof. Using the algorithm of Lemma 4, compute the Tukey depth of q and the closed halfplane h^+ realizing it, where q lies on the line h bounding h^+ . This takes $O(n + k \log k)$ time. By translation and rotation, we can assume that q is the origin and h is the x -axis. Let $P^+ = P \cap h^+$ be the k points realizing the Tukey depth of q , and let $P^- = P \setminus P^+$ be the set of points below the x -axis, so that $|P^-| = n - k \geq 2k$.

Consider the counterclockwise order of the points of P^- starting from the negative side of the x -axis. Let T be the set containing the first and last k points in this order, computed in $O(n)$ time by performing median selection twice. Let $T = \{t_1, \dots, t_{2k}\}$ be the points of T sorted in counterclockwise order. Similarly, let $P^+ = \{p_1, \dots, p_k\}$ be the points of P^+ sorted in counterclockwise order starting from the positive side of the x -axis, see Figure B.1.

Let $\Delta_i = \Delta_{p_i t_i t_{k+i}}$, for $i = 1, \dots, k$. We claim that $q \in \Delta_i$ for all i . To this end, let ℓ_i be the line passing through the origin and p_i , and let ℓ_i^+ denote the halfspace it induces to the left of the vector $\overrightarrow{qp_i}$. Then ℓ_i^+ must contain t_i , as otherwise, $|P \cap \ell_i^+| < k$, contradicting the Tukey depth of q . Namely, the segment $p_i t_i$ intersects the negative side of the x -axis. A symmetric argument, applied to the complement halfplane, implies that the segment $t_{k+i} p_i$ intersects the positive side of the x -axis. Then the origin q is contained in Δ_i , as claimed.

Computing these triangles, we have found a log for q of rank k in $O(n + k \log k)$ time. ◀



■ **Figure B.1** Illustration of the proof of Lemma 5.

Near-Linear-Time, Optimal Vertex Cut Sparsifiers in Directed Acyclic Graphs

Zhiyang He

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Jason Li

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Magnus Wahlström

Royal Holloway, University of London, UK

Abstract

Let G be a graph and $S, T \subseteq V(G)$ be (possibly overlapping) sets of terminals, $|S| = |T| = k$. We are interested in computing a *vertex sparsifier* for terminal cuts in G , i.e., a graph H on a smallest possible number of vertices, where $S \cup T \subseteq V(H)$ and such that for every $A \subseteq S$ and $B \subseteq T$ the size of a minimum (A, B) -vertex cut is the same in G as in H . We assume that our graphs are unweighted and that terminals may be part of the min-cut. In previous work, Kratsch and Wahlström (FOCS 2012/JACM 2020) used connections to matroid theory to show that a vertex sparsifier H with $O(k^3)$ vertices can be computed in randomized polynomial time, even for arbitrary digraphs G . However, since then, no improvements on the size $O(k^3)$ have been shown.

In this paper, we draw inspiration from the renowned *Bollobás's Two-Families Theorem* in extremal combinatorics and introduce the use of total orderings into Kratsch and Wahlström's methods. This new perspective allows us to construct a sparsifier H of $\Theta(k^2)$ vertices for the case that G is a DAG. We also show how to compute H in time near-linear in the size of G , improving on the previous $O(n^{\omega+1})$. Furthermore, H recovers the *closest* min-cut in G for every partition (A, B) , which was not previously known. Finally, we show that a sparsifier of size $\Omega(k^2)$ is required, both for DAGs and for undirected edge cuts.

2012 ACM Subject Classification Theory of computation \rightarrow Sparsification and spanners; Theory of computation \rightarrow Fixed parameter tractability; Mathematics of computing \rightarrow Matroids and greedoids

Keywords and phrases graph theory, vertex sparsifier, representative family, matroid

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.52

Related Version *Full Version*: <https://arxiv.org/abs/2011.13485>

Funding *Jason Li*: Supported in part by NSF award CCF-1907820.

1 Introduction

Let $G = (V, E)$ be an unweighted, directed graph, and let $S, T \subseteq V$ be sets of terminals, not necessarily disjoint. In the vertex sparsifier problem, our goal is to construct a smaller graph H , called a *vertex sparsifier*, that preserves the cut structure of S, T in G . More precisely, H includes all vertices in S, T , and for all subsets $A \subseteq S$ and $B \subseteq T$, the size of the minimum vertex cut separating A and B is the same in G and H . Here, we allow the min-cut to contain vertices from A and B ; for other notions of cut sparsifiers from the literature, see related work, below.

A result of Kratsch and Wahlström proved the first bound on the size of a vertex sparsifier that is polynomial in the number of terminals. When S, T have size k , the vertex sparsifier has $O(k^3)$ vertices. Kratsch and Wahlström's main insight is to phrase the problem in terms of constructing representative families on a certain matroid, after which they can appeal to the rich theory on representative families [19, 16]. Their result, also known as the



© Zhiyang He, Jason Li, and Magnus Wahlström;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 52; pp. 52:1–52:14

Leibniz International Proceedings in Informatics



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

cut-covering lemma in the areas of fixed-parameter tractability and kernelization, has led to many new algorithmic developments [13, 12, 9, 10]. Nevertheless, despite the recent surge in applications of the cut-covering lemma, the original bound of $O(k^3)$ has yet to be improved.

In this paper, we introduce the ordered version of the representative family method, and use it to give a sparsifier on $O(k^2)$ vertices in directed acyclic graphs. This matches lower bounds of $\Omega(k^2)$, which we present in Section 4. Furthermore, unlike Kratsch and Wahlström’s result, our new algorithm runs in near-linear time in the size of the graph, and preserves all *closest* min-cuts between subsets of the terminals. In fact, the latter is an important ingredient in the improved running time; see discussion below. We expect that covering closest min-cuts may lead to further applications in the theory of kernelization. The central method we use is the following theorem.

► **Theorem 1.** *Suppose \mathcal{F} is a family of subsets of \mathbb{F}^d for some field \mathbb{F} and for all $B \in \mathcal{F}$, $|B| = s$. Let*

$$\mathcal{A} = \{A \subseteq \mathbb{F}^d \mid |A| \leq r \text{ and } \exists B \in \mathcal{F} \text{ s.t. } A \uplus B \text{ is linearly independent}\}.$$

Fix any ordering σ of \mathcal{F} , namely $\mathcal{F} = \{B_1, B_2, \dots, B_n\}$, and suppose $d \geq r + s$. Then there exists $\mathcal{B} \subseteq \mathcal{F}$, $\mathcal{B} = \{B_{i_1}, B_{i_2}, \dots, B_{i_m}\}$ where $i_1 < i_2 < \dots < i_m$, such that

- (a) *For all $A \in \mathcal{A}$, there exists $B_{i_k} \in \mathcal{B}$ where $A \uplus B_{i_k}$ is independent and for all $j \in [n]$, $j > i_k$, $A \uplus B_j$ is dependent. Note that B_j is not necessarily in \mathcal{B} . Here \uplus denotes disjoint union, which particularly implies that if A and B are not disjoint, then $A \uplus B$ is a multi-set and therefore dependent.*
- (b) *$m \leq \binom{d}{s}$, and we can find \mathcal{B} algorithmically using $O(\binom{d}{s}ns^\omega + \binom{d}{s}^{\omega-1}n)$ field operations over \mathbb{F} (in particular, in a number of operations linear in $|\mathcal{F}|$).*

The condition that $A \uplus B_j$ is required to be dependent only for $j > i_k$, as opposed to $A \uplus B_j$ being dependent for every $j \neq i_k$, recalls the *skew* version of Bollobás’s two-families theorem, proven by Frankl [8].

Technically, this version is equivalent to the weighted version of the representative family method shown by Fomin et al. [6]. In that version, every element $X \in \mathcal{F}$ has a weight $w(X)$, and condition a is replaced by the condition that $w(B_{i_k})$ is maximal among all sets B_j such that $A \uplus B_j$ is independent. Indeed, given σ we can simply use $w(B_j) = j$, and conversely any input where all the weights are distinct enforces a corresponding total ordering on the elements¹. However, the difference in focus between weights and an ordering is significant, as a total element ordering can carry semantic meaning that is obscured when implemented using weights.

Applying Theorem 1 to vertex cut sparsifiers, we obtain the main result of this paper.

► **Theorem 2.** *Given a directed acyclic graph $G = (V, E)$ with terminal sets S, T of size k , we can find a vertex cut sparsifier of G of size $\Theta(k^2)$ algorithmically in time $\tilde{O}((m+n)k^{O(1)})$.*

Our proof initially follows that of Kratsch and Wahlström [13]. Like Kratsch and Wahlström, we compute a “cut-covering set”, i.e., a set $Z \subseteq V(G)$ such that for every $A \subseteq S$ and $B \subseteq T$ there is an (A, B) -min cut C with $C \subseteq Z$. However, Kratsch and Wahlström’s result is based around *essential vertices*, i.e., vertices $v \in V(G)$ that must be included in any

¹ In the first version of this paper, we presented a proof of Theorem 1 that runs in polynomial time. It was later pointed out to us that this theorem is equivalent to Theorem 3.7 in [6], which runs in near-linear time. We thereby refer the readers to the proof in [6], as our proof shares similar underlying ideas with theirs.

cut-covering set. Using the unordered version of Theorem 1 (see Lemma 1.1 of [11]), they compute a set X of $O(k^3)$ vertices which is guaranteed to include all essential vertices in G . They then eliminate one vertex not in X , recompute the representative family, and repeat until exhaustion. This gives a superlinear running time and a final bound of $|Z| = O(k^3)$. This iterative process for computing Z was required because the initial set X could not be guaranteed to contain *all* vertices of any (A, B) -min cut C , unless that min-cut happens to be unique (i.e., consist only of essential vertices).

We use Theorem 1 to improve over this in two ways. By using the additional power of an ordering, we reduce the bound from $|Z| = O(k^3)$ to $|Z| = O(k^2)$ when G is a DAG. Furthermore, instead of covering some arbitrary min-cut for every pair (A, B) , we observe that our construction guarantees that Z contains the unique *closest* (A, B) -min cut to A , i.e., that mincut C for which the set of vertices reachable from A is minimized. This is an interesting consequence that was not previously known, but additionally, it allows us to significantly improve the running time required to compute a cut-covering set. Since the output of Theorem 1 contains all vertices of the closest (A, B) -min cut C for every (A, B) , we can compute a cut-covering set Z with a single application of Theorem 1, eliminating the iterative nature of [13] and improving the running time of the procedure.

Finally, to achieve a linear running time, one more obstacle must be overcome. In order to apply Theorem 1, we need to compute a representation for the matroids underlying the result, known as *gammoids*, in time linear in $n + m$. The usual method for representing gammoids goes via the class of *transversal matroids*, however, this requires taking the inverse of an $n \times n$ matrix. Luckily, we observe that an older construction of Mason [17] can be used to represent gammoids more efficiently over DAGs; see Lemma 10. This allows us to compute Z in time linear in $n + m$, and computing the final sparsifier H is then a simple task.

We also show that there are graphs G with k terminals such that any cut sparsifier H requires $\Omega(k^2)$ vertices, both when H is a directed vertex cut sparsifier for a DAG G , and when H is an undirected edge cut sparsifier for an undirected graph G .

Related work. Several different notions of cut sparsifiers have been considered, as well as vertex sparsifiers for other problems. Vertex cut sparsifiers were first introduced by Moitra [18] in the setting of approximation algorithms; see also [15, 2, 4]. Recently, they have found applications in fast graph algorithms, especially in the dynamic setting [5, 1, 3]. Compared to our setting, many of these results replicate min-cuts only approximately, rather than exactly, and most apply only to undirected edge cuts. On the other hand, in the general case (e.g., when working with edge cuts or when terminals are not deletable), there is an important distinction between parameterizing by the *number of terminals* and the *total terminal capacity*. Our setting, with deletable terminals, corresponds to parameterizing edge cuts by the total capacity of the terminal set. Previous results for this setting include Kratsch and Wahlström [13] discussed above and Chuzhoy [4], as well as recent results on terminal multicut sparsification [20]. By contrast, parameterizing by the number of terminals in a setting where terminals have unbounded capacity makes for a much harder sparsification task, and this is the setting that has been most commonly considered in approximation algorithms. Indeed, it is known that any exact cut sparsifier for k terminals with unbounded capacity needs to have at least exponential size in k , and possibly double-exponential [14].

2 Preliminaries

Throughout the paper, all graphs are directed and unweighted. We begin with standard terminology on cuts and cut sparsifiers.

► **Definition 3** (Vertex Cut). *Given a directed unweighted graph $G = (V, E)$ with sets $X, Y \subseteq V$, a set $C \subseteq V$ is a vertex cut of (X, Y) if after removing C from G , there does not exist a path from a vertex in X to a vertex in Y . We denote the size of a minimum vertex cut between X, Y in G as $\text{mincut}_G(X, Y)$.*

► **Definition 4** (Vertex Cut Sparsifier). *Consider a directed unweighted graph $G = (V, E)$ with sets $S, T \subseteq V$. A directed unweighted graph $H = (V', F)$ is a vertex cut sparsifier of G if*

- (a) $S, T \subseteq V'$.
- (b) For all $X \subseteq S, Y \subseteq T$, $\text{mincut}_G(X, Y) = \text{mincut}_H(X, Y)$.

The problem we consider in this paper is the minimum size of a vertex sparsifier.

► **Problem 5** (Minimum Vertex Cut Sparsifier). *Given a directed unweighted graph $G = (V, E)$ with terminal sets $S, T \subseteq V$, what is the minimum number of vertices in a vertex cut sparsifier of G ?*

Kratsch and Wahlström [13] obtained a bound for this problem of $O(k^3)$ vertices, where $|S| = |T| = k$. Their application was in the fixed-parameter tractability setting, specifically in constructing kernels for cut-based problems. Our proof utilizes similar matroid-theoretic techniques as in their work. We introduce these techniques next.

► **Definition 6** (Matroid). *Given a finite ground set E , a set system $M = (E, I)$ where $I \subseteq \mathcal{P}(E)$ is called a matroid if*

- (a) $\emptyset \in I$.
- (b) For $X, Y \subseteq E$, if $Y \in I$ and $X \subseteq Y$, then $X \in I$.
- (c) If $X, Y \in I$ and $|X| < |Y|$, then there exists $y \in Y \setminus X$ such that $X \cup \{y\} \in I$.

Central to our proof is the use of gammoids and their representations.

► **Definition 7** (Gammoid). *Given a graph $G = (V, E)$ and a subset of vertices S (which we refer to as the “source vertices”), the gammoid on S is the matroid $M = (V, I)$ where I contains all subsets $T \subseteq V$ such that there exist $|T|$ vertex-disjoint paths from S to T in G .*

► **Definition 8** (Matroid disjoint union). *Given two matroids on disjoint ground sets, their matroid disjoint union is the matroid whose ground set is the union of their ground sets, and a set is independent if the corresponding part in each matroid is independent.*

► **Definition 9** (Representable matroid). *Given a field F , a matroid $M = (E, I)$ is representable over F if there exists a matrix A over the field F and a bijective mapping from E to the columns of A , such that a set $S \subseteq E$ is independent if and only if its corresponding set of columns of A are linearly independent.*

In particular, it is well known in matroid theory that gammoids are representable in randomized polynomial time; see Marx [16]. However, to control the running time, we revisit an older representation by Mason [17], and note that it leads to a representation in near-linear time in $|V| + |E|$ on DAGs. (Proofs of results marked ★ are deferred to the full version of this paper.)

► **Lemma 10** (★ Construction of Gammoid Representation on DAGs). *Given a directed acyclic graph $G = (V, E)$, a set $S \subseteq V$, and $\varepsilon > 0$, with $|V| = n$, $|E| = m$ and $|S| = k$, a representation of the gammoid on S of dimension k over a finite field with entries of bit length $\ell = O(k \log n + \log(1/\varepsilon))$ can be constructed in randomized time $\tilde{O}((n+m)k\ell)$ with one-sided error at most ε , where \tilde{O} hides factors logarithmic in ℓ .*

We also note that the disjoint union of two representable matroids is representable. Given these definitions, we now present the main arguments of this paper.

3 Vertex Cut Sparsifier for DAGs

In this section, we prove our main result, Theorem 2. We first borrow the following key concepts from Kratsch and Wahlström [13].

► **Definition 11** (Essential Vertex). *A vertex $v \in V \setminus (S \cup T)$ is called essential if there exist $X \subseteq S, Y \subseteq T$ such that v belongs to every minimum vertex cut between X, Y .*

► **Definition 12** (Neighborhood Closure). *For a digraph $G = (V, E)$ and a vertex $v \in G$, the neighborhood closure operation is defined by removing v from G and adding an edge from every in-neighbor of v to every out-neighbor of v . The new graph is denoted by $\text{cl}_v(G)$.*

► **Definition 13** (Closest Set). *For sets of vertices $X, A \subseteq V$, A is closest to X if A is the unique min-cut between X and A .*

We introduce the following definitions to simplify our discussions.

► **Definition 14**. *For sets $X \subseteq S, Y \subseteq T$, let C be a vertex cut for X, Y . Let G' be the subgraph formed by the union of all paths from X to Y . The left-hand side of C , denoted $L(C)$, is the set of vertices that are still reachable from X in G' after C is removed. Similarly, the right-hand side of C , denoted $R(C)$, is the set of vertices that can still reach Y in G' after C is removed.*

► **Definition 15** (Saturation). *Let C be a vertex cut for $X \subseteq S, Y \subseteq T$. For a vertex $v \in C$, we say that (C, v) is saturated by X if there exist $|C| + 1$ paths from X to C that are vertex disjoint except for two paths that both ends at v . Similarly, (C, v) is saturated by Y if there exist $|C| + 1$ paths from C to Y that are vertex disjoint except for two paths that both start at v .*

The following three lemmas follow from [13, Section 5.1]. Their proofs, as presented in [13] and Chapter 11.6 of [7], are included in the Appendix for completeness.

► **Lemma 16** (Essential Vertex Lemma). *Let v be an essential vertex with respect to $X \subseteq S, Y \subseteq T$. Let C be any minimum vertex cut between X, Y . Then (C, v) is saturated by both X and Y .*

► **Lemma 17** (Closure Lemma). *If $v \in V \setminus (S \cup T)$ is not an essential vertex, then $\text{cl}_v(G)$ is a vertex cut sparsifier of G .*

► **Lemma 18** (Closest Cut Lemma). *Let C be a vertex cut for $X \subseteq S, Y \subseteq T$ that is closest to X (resp. Y), then for all vertices $v \in C$, (C, v) is saturated by X (resp. Y).*

Using the saturation properties of essential vertices (as in Lemma 16), Kratsch and Wahlström presented a construction of matroids which, combined with the unordered version of Theorem 1, computes a set of vertices P of size $O(k^3)$ that contains all the essential vertices.

They then apply Lemma 17 iteratively to any one vertex not in P and repeat the process. The repetition is required since Lemma 17 may change the essential vertices of the graph.

In our proof, instead of marking only essential vertices we consider all vertices on closest min-cuts. These have weaker saturation properties than essential vertices (as in Lemma 18), but these weaker properties suffice thanks to the ordering imposed in Theorem 1. This ordering, when applied to the topological ordering of a DAG, allows us to mark all vertices of closest min-cuts in a single application of Theorem 1, which eliminates the iterative nature of the previous result. This intuition will be made clear in the following discussion.

We now present our construction, which results in Theorem 2.

► **Theorem 19.** *For a directed acyclic graph $G = (V, E)$ with terminal sets S and T , let $k = |S \cup T|$. Then there exists a set of vertices P of size $O(k^2)$ such that for each pair of $X \subseteq S, Y \subseteq T$, the min- (X, Y) cut that is closest to Y is contained in P . This set can be found in time $\tilde{O}(nk^{2\omega-1} + mk^2)$, and a sparsifier on P can then be constructed in the same asymptotic running time.*

Proof. Let $G_R = (V, E_R)$ be the graph G with the direction of all edges reversed. We make the following modification to our graphs G, G_R . For each vertex $v \in V \setminus (S \cup T)$, add a vertex v' into V and for each directed edge $(u, v) \in E$, add (u, v') into E . We refer to v' as the sink-only copy of v . Denote this new directed graph $G' = (V', E')$, we add sink-only copies of vertices to G_R and obtain $G'_R = (V', E'_R)$. Note that G', G'_R are both acyclic. Enumerate V in a reverse topological ordering, namely $V = (v_1, v_2, \dots, v_n)$ where v_i cannot reach any v_j for $j > i$.

Let $M_1 = (E_1, I_1)$ be the gammoid constructed on the graph G and the set of terminals S in G , and let $M_2 = (E_2, I_2)$ be the gammoid constructed on the graph G'_R and the set of terminals T in G'_R . To distinguish between vertices of E_1 and E_2 , we label vertices in E_1 as v_1, \dots, v_n , and elements in E_2 as $\bar{v}_1, \dots, \bar{v}_n, \bar{v}'_1, \dots, \bar{v}'_n$. Note that E_1 does not contain sink-only copies. For any set of vertices $U \subseteq V$, denote the respective sets in E_1 and E_2 as U_1 and U_2 . Let M be the disjoint union of matroids M_1 and M_2 , so M is representable and it has rank $O(k)$.

Observe that for any $X \subseteq S, Y \subseteq T$, the min-cuts between X and Y in G are the same as in G' because the vertex copies v' we added to G have no outgoing edges. Therefore, G and G' have the same set of closest cuts. We now consider the following constructions. For a min-cut C between X, Y that is closest to Y , and a vertex $v \in C$, define

$$A_{(C,v)} = [(S_1 \setminus X_1) \cup (C_1 \setminus \{v\})] \cup [(T_2 \setminus Y_2) \cup C_2].$$

Let $\tilde{\mathcal{A}}$ consist of $A_{(C,v)}$ for all such cut-vertex pairs. Define

$$\mathcal{F} = \{B_v = \{v, v'\} \mid v \in V\}.$$

We pause for a moment to explain the ideas behind these definitions. First of all, note that the algorithm in Theorem 1 only takes as input a family \mathcal{F} . Intuitively, one should think of the sets in \mathcal{F} as answers to potential queries, and the sets in \mathcal{A} (defined in Theorem 1) as all queries answered by \mathcal{F} . As we will show, the family of queries $\tilde{\mathcal{A}}$ we defined is a subset of \mathcal{A} . If we can further show that for each v in a closest min-cut C , B_v is the unique answer that the algorithm will find for query $A_{(C,v)}$, then we know that the output of the algorithm must contain all vertices of all closest min-cuts, because all queries in $\mathcal{A} \supseteq \tilde{\mathcal{A}}$ must be answered.

More specifically, for each query $A_{(C,v)}$, Theorem 1 promises to find the answer B_u to $A_{(C,v)}$ (which means $A_{(C,v)} \uplus B_u$ is independent in M) that is the last answer according to the reverse topological ordering on \mathcal{F} . This means for all $w > u$, $A_{(C,v)} \uplus B_w$ is dependent

in M . Since our goal is for Theorem 1 to output a set containing all vertices on all closest min-cuts, we want to show that B_v is the last answer in the ordering to $A_{(C,v)}$. This is precisely captured in the following claim.

▷ **Claim 20.** For each $A = A_{(C,v)}$, B_v is the unique set in \mathcal{F} such that

- $A \uplus B_v$ is independent in M , and
- for all $u > v$ in the reverse topological ordering of V , $A \uplus B_u$ is dependent in M .

Proof. We first show that $A \uplus B_v$ is independent. Since M is a disjoint union matroid, we need to show that $(A \cap E_1) \uplus \{v\} = (S_1 \setminus X_1) \cup C_1$ is independent in M_1 and $(A \cap E_2) \uplus \{\bar{v}'\} = (T_2 \setminus Y_2) \cup C_2 \cup \{\bar{v}'\}$ is independent in M_2 .

Since both M_1 and M_2 are gammoids, we need to show existence of vertex disjoint paths from S_1 to $(S_1 \setminus X_1) \cup C_1$. First note that singleton paths can cover all vertices in $S_1 \setminus X_1$. Since C_1 is a min-cut between X_1 and Y_1 , by duality there exist vertex disjoint paths from X_1 to C_1 . Therefore $(S_1 \setminus X_1) \cup C_1$ is independent in M_1 . Similarly, singleton paths can cover all vertices in $T_2 \setminus Y_2$. It suffices for us to show the existence of vertex disjoint paths from Y_2 to $C_2 \cup \{\bar{v}'\}$.

By Lemma 18, there exist $|C_2| + 1$ paths from Y_2 to C_2 that are vertex disjoint except for two paths that both ends at \bar{v} . Therefore, we can redirect one of these two paths to end at \bar{v}' , and we obtain $|C_2| + 1$ vertex disjoint paths from Y_2 to $C_2 \cup \{\bar{v}'\}$. This proves independence.

Now fix $u > v$ in the reverse topological ordering, so that there does not exist a path from v to u . We want to show that either $(A \cap E_1) \uplus \{u\}$ is dependent in M_1 , or $(A \cap E_2) \uplus \{\bar{u}'\}$ is dependent in M_2 . Consider four possible cases:

- u is not on any path from X to Y . Assume for the sake of contradiction that both $(A \cap E_1) \uplus \{u\}$ and $(A \cap E_2) \uplus \{\bar{u}'\}$ are independent. Then there must exist a path from X to u and a path from u to Y , which forms a path from X to Y through u (since G is acyclic), contradiction.
- $u \in L(C)$ (see Definition 14). Then any path from u to Y (or from Y to u in G_R) must intersect with C , which means there does not exist vertex disjoint paths from Y to $C \cup \{u\}$ in G_R . Therefore $(A \cap E_2) \uplus \{\bar{u}'\}$ is dependent.
- $u \in R(C)$. Then any path from X to u must intersect C . Assume for the sake of contradiction that $(A \cap E_1) \uplus \{u\}$ is independent, then there exist vertex disjoint paths from X to $C \setminus \{v\} \cup \{u\}$, which means there is a path from X to u that goes through v . However, since $u > v$ in the topological ordering, there does not exist paths from v to u . This is a contradiction, so $(A \cap E_1) \uplus \{u\}$ is dependent.
- $u \in C$. Then $u \in (A \cap E_1)$, which implies $(A \cap E_1) \uplus \{u\}$ contains two copies of u . Therefore it is dependent.

This completes the proof. ◁

We now apply Theorem 1 on \mathcal{F} with a reverse topological ordering, and we note that the family $\tilde{\mathcal{A}}$ we defined in this proof is a subfamily of \mathcal{A} defined in Theorem 1. Let P be the collection of vertices that Theorem 1 finds. Then by the above claim, for each pair of $X \subseteq S, Y \subseteq T$ and their min-cut C closest to Y , all vertices in C must be in P . Note that this also implies that all essential vertices are in P .

To construct the final sparsifier H on P , for each vertex $u \in P$, we run a depth-first search starting at u on the graph G_u , defined to be G minus the out-edges of vertices in $P \setminus \{u\}$. For each vertex $v \in P \setminus \{u\}$ that is reachable from u in G_u , we add an edge (u, v) to H . Note that this procedure returns the same graph H as the one that sequentially applies Lemma 17 on all vertices not in P , but achieves a shorter runtime of $O(k^2m)$. To see the

equivalence, observe that in both cases, there is an edge (u, v) in the final sparsifier if and only if there is a path from u to v in G whose internal vertices are disjoint from P . We conclude that the output graph H is a valid sparsifier.

For the final running time, note that computing the gammoids takes time $\tilde{O}((m+n)k^2)$ by Lemma 10 with $\varepsilon = 1/n^{O(1)}$, and computing the representative sets takes time $\tilde{O}(nk^{2\omega-1})$ by taking $d = k$ and $s = 2$ in Theorem 1. \blacktriangleleft

We make a few remarks regarding this proof. Intuitively, the gammoid M_2 and its respective query $A_{(C,v)} \cap E_2 = (T_2 \setminus Y_2) \cup C_2$ is used to filter out all vertices on the left-hand side of C , because no vertex in $L(C)$ can reach T without crossing C . The gammoid M_1 and its query $A_{(C,v)} \cap E_1 = (S_1 \setminus X_1) \cup (C_1 \setminus \{v\})$, however, is different, because the query itself allows terminals in X to reach the right-hand side of C through v . If we do not impose a reverse topological ordering on the vertices and use the unordered version of Theorem 1 (Lemma 1.1 in [11]), our proof will fail – there may exist $u \in R(C)$ reachable from v such that $A_{(C,v)} \uplus B_u$ is independent, and the algorithm may not find B_v as the answer. By imposing the reverse topological ordering, we demand the algorithm to find the answer last in the ordering, thereby ensuring the algorithm discovering B_v .

This is the critical difference between our proof and Kratsch and Wahlström’s proof. In Kratsch and Wahlström’s paper, they presented a construction with three matroids – one gammoid on G' and S_1 (note that our first gammoid is constructed on G and S_1), one gammoid on G'_R and T_2 , and one uniform matroid of rank k on V . They then utilized the property that essential vertices can be saturated from both sides (see Lemma 16) and constructed queries similar to our definition of $A_{(C,v)}$. Their first (resp. second) gammoid serves to filter out $R(C)$ (resp. $L(C)$), and they use the uniform matroid to filter out vertices $u \neq v \in C$. We managed to merge their first gammoid and uniform matroid with an asymmetrical construction, thereby proving a stronger bound.

We finally remark that in our proof, we never explicitly compute the queries $A_{(C,v)}$. As mentioned previously, the algorithm in Theorem 1 only takes as input the family \mathcal{F} , and we are simply showing that given our construction of \mathcal{F} , $\tilde{\mathcal{A}}$ is a subset of queries answered. If one can show that another meaningful set of queries are answered by the same \mathcal{F} , then they can derive more properties of the output set of the algorithm (while the output set itself remains unchanged).

We now present tight lower bound constructions in the following section.

4 Lower Bound Constructions

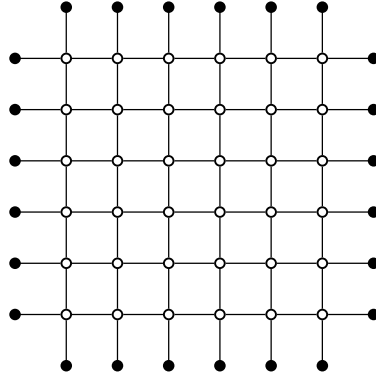
In this section we present two constructions graphs for which any vertex cut sparsifier requires $\Omega(k^2)$ vertices. The first construction is presented by Kratsch and Wahlström [13] (construction found in arXiv preprint only); note that the graph is a DAG.

► **Lemma 21 (★).** *Let S and T be two vertex sets of size $2k$. Enumerate them as*

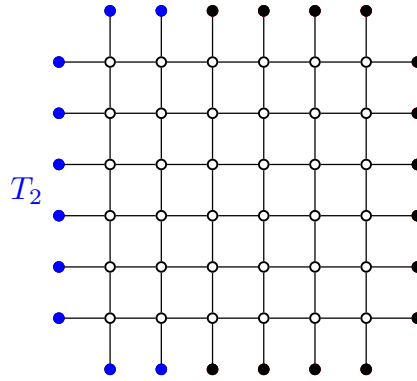
$$S = \{v_1, v'_1, v_2, v'_2, \dots, v_k, v'_k\}, \quad T = \{u_1, u'_1, u_2, u'_2, \dots, u_k, u'_k\}.$$

For each $i, j \in [k]$, create a vertex $w_{i,j}$ and add edges from v_i, v'_i to $w_{i,j}$, and from $w_{i,j}$ to u_j, u'_j . Any vertex cut sparsifier for the resulting graph requires $\Omega(k^2)$ vertices.

Next, we show the same bound holds also for *edge cut* sparsifiers, when both G and its sparsifier H are undirected. For simplicity, we also restrict ourselves to just one terminal set T .



■ **Figure 1** The lower bound construction for $k = 6$.



■ **Figure 2** The set T_i for $i = 2$.

► **Definition 22** ((Undirected) Edge Cut Sparsifier). *Consider an unweighted, undirected graph $G = (V, E)$ with terminal set $T \subseteq V$. An unweighted, undirected graph $H = (V', F)$ is an edge cut sparsifier of G if*

- (a) $T \subseteq V'$.
- (b) For all disjoint $X, Y \subseteq T$, $\text{emincut}_G(X, Y) = \text{emincut}_H(X, Y)$.

Consider the k -by- k grid with leaf terminals attached in Figure 1: begin with a k -by- k grid of non-terminals, and add the following leaf (degree-1) terminals:

1. one leaf terminal adjacent to each vertex on the top row, called the *top* terminals,
 2. one leaf terminal adjacent to each vertex on the bottom row, called the *bottom* terminals,
 3. one leaf terminal adjacent to each vertex on the left column, called the *left* terminals, and
 4. one leaf terminal adjacent to each vertex on the right column, called the *right* terminals.
- Note that non-terminals on the corner of the grid have two terminals attached. This concludes the construction of the terminal set T . We now show that any edge cut sparsifier of G , even with directed edges allowed, has at least k^2 vertices. Since G has $O(k)$ terminals, this proves the quadratic lower bound.

► **Lemma 23.** *Any (undirected) edge cut sparsifier of G has $\Omega(k^2)$ vertices.*

For the rest of this section, we prove Lemma 23. For an undirected graph G and a subset of vertices S , we define $\partial_G S$ as the set of edges with exactly one endpoint in S .

Consider a directed sparsifier H with vertex set $V' \supseteq T$. For $0 \leq i \leq k$, let $T_i \subseteq T$ be following set of terminals: all of the left terminals, plus the first i top and bottom terminals, counting from the left (see Figure 2). Let $C_i \subseteq V'$ be T_i 's side of the mincut between $T_i, T \setminus T_i$

in the sparsifier H , which must have $|\partial C_i| = |\partial(V' \setminus C_i)| = k$, since $\text{emincut}_G(T_i, T \setminus T_i)$ is k and H is a sparsifier of G . Similarly, let $T'_i \subseteq T$ be all the top terminals, plus the first i left and right terminals, counting from the top, and let $R_i \subseteq V'$ be T'_i 's side of the mincut between $T'_i, T \setminus T'_i$ in H , so that $|\partial R_i| = |\partial(V' \setminus R_i)| = k$.

▷ **Claim 24.** Without loss of generality, we may assume that $C_0 \subseteq C_1 \subseteq \dots \subseteq C_k$ and $R_0 \subseteq R_1 \subseteq \dots \subseteq R_k$.

Proof. We only prove the statement for C_i ; the one for R_i follows by symmetry of G . Suppose for contradiction that $C_i \setminus C_{i+1} \neq \emptyset$. By submodularity,

$$|\partial C_i| + |\partial C_{i+1}| \geq |\partial(C_i \cap C_{i+1})| + |\partial(C_i \cup C_{i+1})|.$$

Since $C_i \cap C_{i+1}$ is a $(T_i, T \setminus T_i)$ -cut in H and $C_i \cup C_{i+1}$ is a $(T_{i+1}, T \setminus T_{i+1})$ -cut in H , their cut values $\partial(C_i \cap C_{i+1})$ and $\partial(C_i \cup C_{i+1})$ are at least k . Therefore,

$$k + k = |\partial C_i| + |\partial C_{i+1}| \geq |\partial(C_i \cap C_{i+1})| + |\partial(C_i \cup C_{i+1})| \geq k + k,$$

so the inequality must be an equality. It follows that we can replace C_i with $C_i \cap C_{i+1}$, which is still a $(T_i, T \setminus T_{i+1})$ -cut in H , and we can also replace C_{i+1} with $C_i \cup C_{i+1}$. While there exists an i such that $C_i \setminus C_{i+1} \neq \emptyset$, we perform the replacement; this can only happen a finite number of times, since the quantity $\sum_{i=0}^k |C_i|^2$ increases by at least 1 each time and has an upper limit. ◁

For each $1 \leq i, j < k$, define $S_{i,j} \subseteq V'$ as $S_{i,j} = (C_{i+1} \setminus C_i) \cap (R_{j+1} \setminus R_j)$; see Figure 3a. Our goal is to prove that $S_{i,j} \neq \emptyset$ for all i, j ; since the sets are disjoint over all $1 \leq i, j \leq k$, this implies the k^2 bound.

▷ **Claim 25.** There are no edges between $S_{i,j}$ and $S_{i',j'}$ for $i \neq i'$ and $j \neq j'$.

Proof. First, consider some $1 \leq i, i', j, j' \leq k$ where $i < i'$ and $j < j'$ (see Figure 3a). By submodularity on the sets C_{i+1} and $R_{j'+1}$,

$$|\partial C_{i+1}| + |\partial R_{j'+1}| \geq |\partial(C_{i+1} \cap R_{j'+1})| + |\partial(C_{i+1} \cup R_{j'+1})|.$$

Since $\partial(C_{i+1} \cap R_{j'+1})$ is a $(T_{i+1} \cap T'_{j'+1}, T \setminus (T_{i+1} \cap T'_{j'+1}))$ -cut, its value is at least

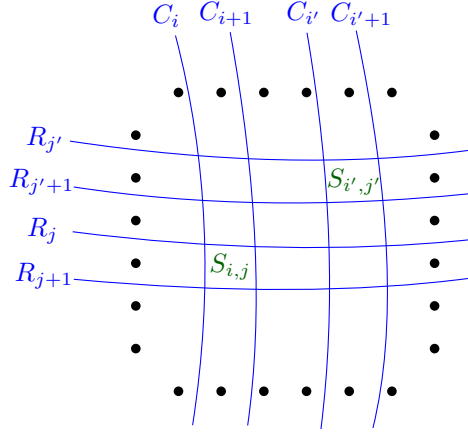
$$\text{emincut}_G(T_{i+1} \cap T'_{j'+1}, T \setminus (T_{i+1} \cap T'_{j'+1})) = (i+1) + (j'+1).$$

Similarly, $|\partial(C_{i+1} \cup R_{j'+1})| \geq k - (i+1) + k - (j'+1)$. It follows that

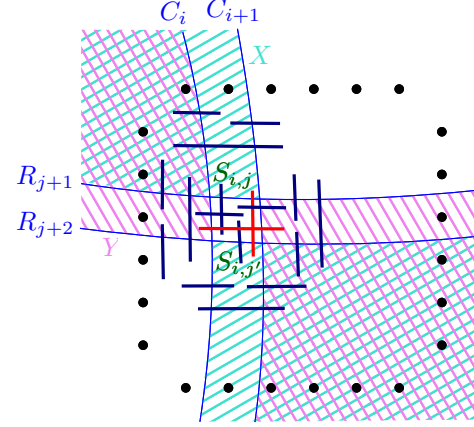
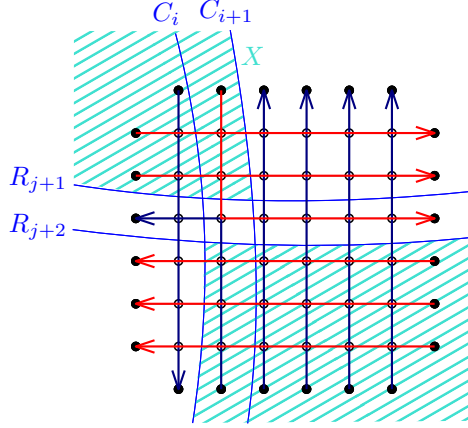
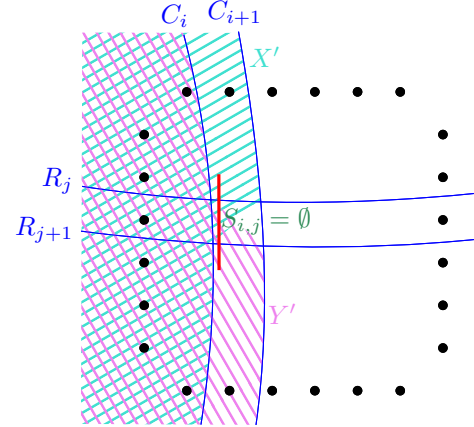
$$\begin{aligned} k + k &= |\partial C_{i+1}| + |\partial R_{j'+1}| \geq |\partial(C_{i+1} \cap R_{j'+1})| + |\partial(C_{i+1} \cup R_{j'+1})| \\ &\geq (i+1) + (j'+1) + k - (i+1) + k - (j'+1) \\ &= k + k, \end{aligned}$$

so the inequality must be an equality. It follows that there are no edges between $C_{i+1} \setminus R_{j'+1}$ and $R_{j'+1} \setminus C_{i+1}$, since those edges would make the submodularity inequality strict. In particular, there are no edges between $S_{i,j}$ and $S_{i',j'}$.

Finally, the $i < i'$ and $j < j'$ assumptions can be removed essentially by symmetry, replacing C_{i+1} by $V' \setminus C_{i+1}$ or $R_{j'+1}$ by $V' \setminus R_{j'+1}$ (or both). ◁



(a) The setting for the proof of Claim 25.

(b) The definitions of X and Y , and the different types of edges that are allowed by Claim 25 and leave or enter the sets R_{j+1} , R_{j+2} , C_i , C_{i+1} . The long edges marked in red make the inequality $|\partial_H X| + |\partial_H Y| \leq |\partial_H R_{j+1}| + |\partial_H R_{j+2}| + |\partial_H C_i| + |\partial_H C_{i+1}|$ strict.(c) The cut and flow that establishes that $\text{emincut}_G(X \cap T, T \setminus X) = 2k$. The flow is drawn in red and dark blue.(d) The definitions of X' and Y' . The red edge makes the submodularity inequality strict.■ **Figure 3** Figures for Lemma 23.

▷ **Claim 26.** There are no edges between $S_{i,j}$ and $S_{i,j'}$ for $|j - j'| \geq 2$. Similarly, there are no edges between $S_{i,j}$ and $S_{i',j}$ for $|i - i'| \geq 2$.

Proof. We first prove the inequality for $S_{i,j}$ and $S_{i,j'}$ for $j' \geq j + 2$. As shown in Figure 3b, define $X = (R_{j+1} \cap C_{i+1}) \cup ((V \setminus R_{j+2}) \cap (V \setminus C_i))$ and $Y = (R_{j+2} \cap C_i) \cup ((V' \setminus R_{j+1}) \cap (V' \setminus C_{i+1}))$. We now claim the inequality

$$|\partial_H X| + |\partial_H Y| \leq |\partial_H R_{j+1}| + |\partial_H R_{j+2}| + |\partial_H C_i| + |\partial_H C_{i+1}| \quad (1)$$

by examining each type of edge in Figure 3b and its contribution to both sides of the inequality:

1. Each edge between $C_i \cap R_{j+1}$ and $(C_{i+1} \setminus C_i) \cap R_{j+1}$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H C_i|$.
2. Each edge between $C_{i+1} \cap R_{j+1}$ and $(V \setminus C_{i+1}) \cap R_{j+1}$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H C_{i+1}|$.

3. Each edge between $C_i \cap R_{j+1}$ and $(V \setminus C_{i+1}) \cap R_{j+1}$ contributes 1 to $|\partial_H X|$ and $|\partial_H Y|$ and 1 to $|\partial_H C_i|$ and $|\partial_H C_{i+1}|$.
4. Each edge between $C_i \cap R_{j+1}$ and $C_i \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H R_{j+1}|$.
5. Each edge between $C_i \cap R_{j+2}$ and $C_i \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H R_{j+1}|$.
6. Each edge between $C_i \cap R_{j+1}$ and $C_i \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and $|\partial_H Y|$ and 1 to $|\partial_H R_{j+1}|$ and $|\partial_H R_{j+2}|$.
7. Each edge between $C_i \cap (V \setminus R_{j+2})$ and $(C_{i+1} \setminus C_i) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H C_i|$.
8. Each edge between $C_{i+1} \cap (V \setminus R_{j+2})$ and $(V \setminus C_{i+1}) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H C_{i+1}|$.
9. Each edge between $C_i \cap (V \setminus R_{j+2})$ and $(V \setminus C_{i+1}) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and $|\partial_H Y|$ and 1 to $|\partial_H C_i|$ and $|\partial_H C_{i+1}|$.
10. Each edge between $(V \setminus C_{i+1}) \cap R_{j+1}$ and $(V \setminus C_{i+1}) \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H R_{j+1}|$.
11. Each edge between $(V \setminus C_{i+1}) \cap R_{j+2}$ and $(V \setminus C_{i+1}) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H R_{j+1}|$.
12. Each edge between $(V \setminus C_{i+1}) \cap R_{j+1}$ and $(V \setminus C_{i+1}) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and $|\partial_H Y|$ and 1 to $|\partial_H R_{j+1}|$ and $|\partial_H R_{j+2}|$.
13. Each edge between $C_i \cap (R_{j+2} \setminus R_{j+1})$ and $(C_{i+1} \setminus C_i) \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H C_i|$.
14. Each edge between $(C_{i+1} \setminus C_i) \cap (R_{j+2} \setminus R_{j+1})$ and $(V \setminus C_{i+1}) \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H Y|$ and 1 to $|\partial_H C_{i+1}|$.
15. Each edge between $C_i \cap (R_{j+2} \setminus R_{j+1})$ and $(V \setminus C_{i+1}) \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H C_i|$ and 1 to $|\partial_H C_{i+1}|$.
16. Each edge between $(C_{i+1} \setminus C_i) \cap R_{j+1}$ and $(C_{i+1} \setminus C_i) \cap (R_{j+2} \setminus R_{j+1})$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H R_{j+1}|$.
17. Each edge between $(C_{i+1} \setminus C_i) \cap (R_{j+2} \setminus R_{j+1})$ and $(C_{i+1} \setminus C_i) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H X|$ and 1 to $|\partial_H R_{j+2}|$.
18. Each edge between $(C_{i+1} \setminus C_i) \cap R_{j+1}$ and $(C_{i+1} \setminus C_i) \cap (V \setminus R_{j+2})$ contributes 1 to $|\partial_H R_{j+1}|$ and 1 to $|\partial_H R_{j+2}|$.

All of the above types of edges contribute the same to both sides of (1) except those of type 15 and 18, namely those indicated by long red edges in Figure 3b. We call such edges *red*. Then, the inequality (1) is strict if and only if red edges are present.

Since all edges between $S_{i,j}$ and $S_{i',j'}$ are red, it suffices to show that (1) is actually an equality, which would exclude all red edges and hence all edges between $S_{i,j}$ and $S_{i',j'}$ as well. Observe that $\text{emincut}_G(X \cap T, T \setminus X) = 2k$, as seen in Figure 3c, which shows a cut and a flow both of value $2k$. Similarly, $\text{emincut}_G(Y \cap T, T \setminus Y) = 2k$. Since H is a sparsifier of G , we must have

$$\begin{aligned} 2k + 2k &= |\partial_H X| + |\partial_H Y| \leq |\partial_H R_{j+1}| + |\partial_H R_{j+2}| + |\partial_H C_i| + |\partial_H C_{i+1}| \\ &= k + k + k + k. \end{aligned}$$

In other words, the inequality is tight, as desired.

The case for $S_{i,j}$ and $S_{i',j'}$ for $i' \geq i+2$ is similar. Note that edges between $S_{i,j}$ and $S_{i',j'}$ correspond to the red horizontal edges in Figure 3b (for different values of i, j), which we have already shown do not exist. \triangleleft

▷ Claim 27. For each $1 \leq i, j \leq k$, we have $S_{i,j} \neq \emptyset$.

Proof. Suppose for contradiction that $S_{i,j} = \emptyset$ for some $1 \leq i, j \leq k$. As shown in Figure 3d, define the sets $X' = C_i \cup (C_{i+1} \cap R_j)$ and $Y' = C_i \cup (C_{i+1} \setminus R_{j+1})$. We have $X' \cap Y' = C_i$, and since $S_{i,j} = \emptyset$ by assumption, we also have $X' \cup Y' = C_{i+1}$. By submodularity,

$$|\partial_H X'| + |\partial_H Y'| \geq |\partial_H(X' \cap Y')| + |\partial_H(X' \cup Y')| = |\partial_H C_i| + |\partial_H C_{i+1}|.$$

Moreover, the inequality is tight because the only types of edges that can make the inequality strict (marked red in Figure 3d) are prohibited by Claim 26. Since H is a sparsifier of G ,

$$\text{emincut}_G(X' \cap T, T \setminus X') + \text{emincut}_G(Y' \cap T, T \setminus Y') \leq |\partial_H X'| + |\partial_H Y'| = |\partial_H C_i| + |\partial_H C_{i+1}| = 2k.$$

However, it is not hard to see that $\text{emincut}_G(X' \cap T, T \setminus X') = \text{emincut}_G(Y' \cap T, T \setminus Y') = k+1$, a contradiction. ◁

5 Conclusions

We showed that every unweighted, directed acyclic graph G with k terminals admits a vertex cut sparsifier H with $O(k^2)$ vertices, assuming that the terminals are deletable. This improves the previous result by Kratsch and Wahlström of $O(k^3)$ vertices, for general directed graphs [13]. Furthermore, the sparsifier can be computed in near-linear time in the size of G , specifically in time $O((m+n)k^{O(1)})$ plus $O((m+n)k^{O(1)})$ field operations over a finite field with entries of bitlength $O(k \log n)$, where $n = |V(G)|$ and $m = |E(G)|$. This improves over previous work [13], whose time complexity was not explicitly given but is at least $O(n^{\omega+1})$ due to the repeated construction of a representation of a gammoid.

Furthermore, we showed that $\Omega(k^2)$ vertices in a sparsifier may be required, both for vertex cuts in DAGs and for the seemingly simpler setting of undirected edge cuts. However, we leave it open whether such a bound applies to the mixed setting, where we want to preserve undirected edge cuts in the input graph G but allow the sparsifier to be a directed graph.

More importantly, we leave open the question of whether vertex cut sparsifiers of $O(k^2)$ vertices exist for general directed graphs. We conjecture that $O(k^2)$ is the correct bound for general directed graphs, but we were not able to find a proof.

References

- 1 Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1206–1225. SIAM, 2021.
- 2 Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 265–274. IEEE, 2010.
- 3 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science*, pages 1135–1146. IEEE, 2020.
- 4 Julia Chuzhoy. On vertex sparsifiers with Steiner nodes. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 673–688, 2012.
- 5 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 914–925, 2019.

- 6 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.
- 7 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 8 Peter Frankl. An extremal problem for two families of sets. *Eur. J. Comb.*, 3(2):125–127, 1982. doi:10.1016/S0195-6698(82)80025-5.
- 9 Eva-Maria C Hols and Stefan Kratsch. A randomized polynomial kernel for subset feedback vertex set. *Theory of Computing Systems*, 62(1):63–92, 2018.
- 10 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM Journal on Discrete Mathematics*, 32(3):1806–1839, 2018.
- 11 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 450–459. IEEE, 2012.
- 12 Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms (TALG)*, 10(4):1–15, 2014.
- 13 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020.
- 14 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1789–1799. SIAM, 2013.
- 15 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and Lipschitz extendability. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 255–264. IEEE, 2010.
- 16 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.
- 17 John H Mason. On a class of matroids arising from paths in graphs. *Proceedings of the London Mathematical Society*, 3(1):55–74, 1972.
- 18 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 3–12. IEEE, 2009.
- 19 Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.
- 20 Magnus Wahlström. On quasipolynomial multicut-mimicking networks and kernelization of multiway cut problems. In *ICALP*, volume 168 of *LIPICs*, pages 101:1–101:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Closing the Gap for Single Resource Constraint Scheduling

Klaus Jansen 

Universität Kiel, Germany

Malin Rau 

Universität Hamburg, Germany

Abstract

In the problem called single resource constraint scheduling, we are given m identical machines and a set of jobs, each needing one machine to be processed as well as a share of a limited renewable resource R . A schedule of these jobs is feasible if, at each point in the schedule, the number of machines and resources required by jobs processed at this time is not exceeded. It is NP-hard to approximate this problem with a ratio better than $3/2$. On the other hand, the best algorithm so far has an absolute approximation ratio of $2 + \varepsilon$. In this paper, we present an algorithm with absolute approximation ratio $(3/2 + \varepsilon)$, which closes the gap between inapproximability and best algorithm with exception of a negligible small ε .

2012 ACM Subject Classification Theory of computation \rightarrow Packing and covering problems; Theory of computation \rightarrow Scheduling algorithms

Keywords and phrases resource constraint scheduling, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.53

Related Version Full Version: <http://arxiv.org/abs/2107.01613>

Funding Research was supported by German Research Foundation (DFG) project JA 612 / 20-1.

1 Introduction

In the single resource constraint scheduling problem, we are given m identical machines, a discrete renewable resource with a fixed size $R \in \mathbb{N}$ and a set of n jobs \mathcal{J} . Each job has a processing time $p(j) \in \mathbb{Q}$. We define the total processing time of a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ as $p(\mathcal{J}') := \sum_{j \in \mathcal{J}'} p(j)$. To be scheduled, each job $j \in \mathcal{J}$ needs one of the machines as well as a fix amount $r(j) \in \mathbb{N}$ of the resource which it will allocate during the complete processing time $p(j)$ and which it deallocates as soon as it has finished its processing. Neither machine nor any part of the resource can be allocated by two different jobs at the same time. We define the area of a job as $\text{area}(j) := r(j) \cdot p(j)$ and the area of a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ as $\text{area}(\mathcal{J}') := \sum_{j \in \mathcal{J}'} r(j) \cdot p(j)$.

A schedule $\sigma : \mathcal{J} \rightarrow \mathbb{N}$ maps each job $j \in \mathcal{J}$ to a starting point $\sigma(j) \in \mathbb{Q}$. We say a schedule is feasible if

$$\forall t \in \mathbb{Q} : \sum_{j: t \in [\sigma(j), \sigma(j) + p(j))} r(j) \leq R \text{ and} \quad (\text{resource condition})$$

$$\forall t \in \mathbb{Q} : \sum_{j: t \in [\sigma(j), \sigma(j) + p(j))} 1 \leq m. \quad (\text{machine condition})$$

Given a schedule σ where these two conditions hold, we can generate an assignment of resources and machines to the jobs such that each machine and each resource part is allocated by at most one job at a time, see [27]. The objective is to find a feasible schedule, which minimizes the total length of the schedule called makespan, i.e., we have to minimize $\max_{j \in \mathcal{J}} \sigma(j) + p(j)$.



© Klaus Jansen and Malin Rau;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 53; pp. 53:1–53:15



Leibniz International Proceedings in Informatics

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

This problem arises naturally, e.g., in parallel computing, where jobs that are scheduled in parallel share common memory or in production logistics where different jobs need a different number of people working on it. From a theoretical perspective this problem is a sensible generalization of problems like scheduling on identical machines, parallel task scheduling and bin packing with cardinality constraint.

The algorithm with the best absolute ratio so far is a $(2 + \varepsilon)$ -approximation by Niemeier and Wiese [31]. In this paper, we close this gap between approximation and lower bound by presenting an algorithm with approximation ratio $(3/2 + \varepsilon)$.

► **Theorem 1.** *There is an algorithm for single resource constraint scheduling with approximation ratio $(3/2 + \varepsilon)$ and running time $\mathcal{O}(n \log(1/\varepsilon)) + \mathcal{O}(n)(m \log(R)/\varepsilon)^{\mathcal{O}_\varepsilon(1)}$, where \mathcal{O}_ε dismisses all factors solely dependent on $1/\varepsilon$.*

As a by-product of this algorithm, we also present an algorithm, which has an approximation guarantee of $(1 + \varepsilon)\text{OPT} + p_{\max}$. Note that we can scale each instance such that $p_{\max} = 1$ and hence we can see p_{\max} as a constant independent of the instance. Algorithms with an approximation guarantee of the form $(1 + \varepsilon)\text{OPT} + c$ for some constant c are called asymptotic polynomial time approximation schemes (APTAS). Note that this algorithm is a $(2 + \varepsilon)$ -approximation as well, but improves this ratio, in the case that p_{\max} is strictly smaller than OPT and for $p_{\max} < \text{OPT}/2$ improves the approximation ratio of the algorithm from Theorem 1.

► **Theorem 2.** *There is an APTAS for single resource constraint scheduling with an additive term p_{\max} and running time $\mathcal{O}(n \log(1/\varepsilon + n)) + \mathcal{O}(n)(m \log(R)/\varepsilon)^{\mathcal{O}_\varepsilon(1)}$.*

In the schedule generated by this APTAS, almost all jobs are completed before $(1 + \mathcal{O}(\varepsilon))\text{OPT}$, except for a small set \mathcal{J}' of jobs that all start simultaneously at $(1 + \mathcal{O}(\varepsilon))\text{OPT}$, after the processing of all other jobs is finished. The processing of this set \mathcal{J}' causes the additive term p_{\max} .

Methodology and Organization of this Paper

In Section 2, we will present the main results to generate the APTAS from Theorem 2. The general structure of the algorithm can be summarized as follows. First, we simplify the instance by rounding the processing time of the jobs and partitioning them into large, medium, and small corresponding to their processing time. Afterward, we use some linear programming approaches to find a placement of these jobs inside the optimal packing. The few jobs that are placed fractionally with this linear program will be placed on top of the packing contributing to the set \mathcal{J}' which was mentioned before.

As usual for this kind of algorithms for packing and scheduling problems, we divide the jobs into large, medium and small jobs. While for the placement of medium and small jobs, we use techniques already known, see e.g. [21], we used a new technique to place the large jobs. For the following $(3/2 + \varepsilon)$ approximation, it is important to guarantee that only $\mathcal{O}_\varepsilon(1)$, i.e. constant in $1/\varepsilon$, large jobs are not placed inside the optimal scheduling area. To find such a schedule, we divide the schedule into $\mathcal{O}_\varepsilon(1)$ time slots and guess the machine requirement and a rounded resource requirement of large jobs during this slot. Afterward, the large jobs are placed inside this profile using a linear program, which schedules only $\mathcal{O}_\varepsilon(1)$ of these jobs fractionally. We have to remove these $\mathcal{O}_\varepsilon(1)$ fractionally scheduled jobs, as well as only one extra job per time slot due to the rounded guess, and assign them to the set \mathcal{J}' .

Afterward, in Section 3, we present the $(3/2 + \varepsilon)$ -approximation and prove Theorem 1. Instead of placing the fractional jobs on top of the packing, we stretch the packing by $(1/2 + \mathcal{O}(\varepsilon))\text{OPT}$, and place the fractional scheduled large jobs inside a gap in this stretched

schedule. This stretching allows us to define a common finishing point of (almost) all the jobs, which have a processing time larger than $\text{OPT}/2$ and, thus, we avoid scheduling them fractionally with the linear program. While the general idea of creating such a gap was used before, e.g., in [19], the novelty of this approach is to search for this gap at multiple points in time while considering two and not only one constraint. This obstacle of considering two instead of one constraint while searching for a gap requires a more careful analysis of the shifted schedule, as was needed in other gap constructions.

Related Work

The problem resource constraint scheduling is one of the classical problems in scheduling. It was first studied in 1975 by Garey and Graham [10]. Given m identical machines and capacities R_1, \dots, R_s of s distinct resources such that each job requires a share of each of them, they proved that the greedy list algorithm produces a schedule of length at most $(s + 2 - (2s + 1)/m)\text{OPT}$. This corresponds to an approximation ratio of $(3 - 3/m)$ for the case of $s = 1$ i.e., the problem studied in this paper. In the same year Garey and Johnson [11] showed that this general scheduling problem is NP-complete even if just one resource is given, i.e., $s = 1$. Lately, Niemeier and Wiese [31] presented a $(2 + \varepsilon)$ -approximation for single resource constraint scheduling, and this is the best known ratio so far.

Note that the problem single resource constraint scheduling contains multiple problems as subproblems. When all the processing times are equal to one, this problem corresponds to bin packing with cardinality constraint. Hence there is no algorithm with an approximation guarantee better than $3/2$ for this problem unless $P = NP$. For bin packing with cardinality constraint Epstein and Levin [8] presented an AFPTAS. This AFPTAS was improved and extended to work for single resource constraint scheduling by Jansen et al. [21]. It has an additive term of $\mathcal{O}(p_{\max} \log(1/\varepsilon)/\varepsilon)$.

On the other hand, if the number of machines m is larger than n , the constraint that only m jobs can be processed at the same time is no longer a restriction. The resulting problem is known as the parallel task scheduling problem. This problem is strongly NP-complete for $R \geq 4$ [17] and there exists a pseudo polynomial algorithm for $R \leq 3$ [7]. Furthermore, for R constant and $R \in n^{\mathcal{O}(1)}$ there exists polynomial time approximation schemes by Jansen and Porkolab [22] as well as Jansen and Thöle [26] respectively. For an arbitrary large R there exists no algorithm with approximation ratio smaller than $3/2$ unless $P = NP$. The best algorithm for this scenario is a $(3/2 + \varepsilon)$ approximation by Jansen [19].

Finally, if each job requires at most R/m from the resource, the used resources are no longer a restriction. The corresponding problem is known as the NP-hard problem makespan scheduling on identical machines. For this problem several algorithms are known, see, e.g., [13, 18, 2, 20].

An interesting extension of the considered problem is the consideration of resource dependent processing times. In this scenario, instead of a fixed resource requirement for the jobs, each jobs processing time depends on the amount of allocated resources. The first result for this extension was achieved by Grigoriev et al. [14]. They studied a variant where the processing time of a job depends on the machine it is processed on as well as the number of assigned resources and described a 3.75-approximation. For the case of identical machines this result was improved by Kellerer [28] to a $(3.5 + \varepsilon)$ -approximation. Finally, Jansen et al [21] presented an AFPTAS for this problem with additive term $\mathcal{O}(\pi_{\max} \log(1/\varepsilon)/\varepsilon)$, where π_{\max} is the largest occurring processing time considering all possible resource assignments.

Closely related to single resource constraint scheduling is the strip packing problem. Here we are given a set of rectangular items that have to be placed overlapping free into a strip with bounded width and infinite height. We can interpret single resource constraint

scheduling as a Strip Packing problem by setting the jobs processing time to the height of a rectangular item and the resource requirement to its width. The difference to strip packing is now, that when placing an item, we are allowed to slice it vertically as long as the lower border of all slices are placed at the same *vertical level*. Furthermore, we have an single resource constraint scheduling carnality condition that allows only m items to intersect each horizontal line through the strip. The strip packing problem has been widely studied [3, 4, 5, 6, 12, 16, 25, 29, 32, 33, 34, 35]. The algorithm with approximation ratio $(5/3 + \varepsilon)$, which is the smallest so far, was presented by Harren, Jansen, Prädel, and van Stee [15]. On the other hand, using a reduction from the partition problem, we know that there is no polynomial-time algorithm with an approximation ratio smaller than $3/2$. Closing this gap between the best approximation ratio and lower bound represents an open question. Strip packing has also been studied with respect to asymptotic approximation ratio [3, 6, 12, 29]. The best algorithms in this respect are an *AFPTAS* with additive term $\mathcal{O}(1/\varepsilon \log(1/\varepsilon))h_{\max}$ [35, 5] and an APTAS with additive term h_{\max} [25], where h_{\max} is the tallest height in the set of given items. Finally, this problem has been studied with respect to pseudo-polynomial processing time [36, 26, 30, 9, 23, 1], where the width of the strip is allowed to occur polynomial in the running time of the algorithm. There is no pseudo-polynomial algorithm with an approximation ratio smaller than $5/4$ [17] and a ratio of $(5/4 + \varepsilon)$ is achieved by the algorithm in [24].

2 APTAS with additive term p_{\max}

In this section, we present an asymptotic PTAS for the single resource constraint scheduling problem which has an approximation guarantee of $(1 + \varepsilon)\text{OPT} + p_{\max}$ and a running time of $\mathcal{O}(n \log(n)) + (m \log(R))^{\mathcal{O}_\varepsilon(1)}$, i.e., we prove Theorem 2 in this section. Due to space limitations the proofs of this section can be found in the appendix.

Simplifying the input instance. In the first step of the algorithm, we simplify the given instance such that it has a simple structure and a reduced set of processing times. Consider the lower bound on the optimal makespan $T := \min\{p_{\max}, \text{area}(\mathcal{J})/R, p(\mathcal{J})/m\}$. By the analysis of the greedy list schedule, as described in [31], we know that the optimal schedule has a size of at most $\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}) + p_{\max} \leq 4T$, giving us appropriate bounds for a dynamic search framework.

In the next step, we will create a gap between jobs with a large processing time and jobs with a small processing time, by removing a set of medium sized jobs. We want to schedule this set of medium sized jobs in the beginning or end of the schedule using the greedy list schedule. However this schedule of the medium sized items should add at most $\mathcal{O}(\varepsilon)\text{OPT}$ to the makespan. The schedule generated by the greedy list schedule algorithm has a makespan of at most $\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}) + p_{\max} \leq 4\text{OPT}$. Hence, we choose the set of medium jobs \mathcal{J}_M such that $p_{\max}(\mathcal{J}_M) \leq \varepsilon\text{OPT}$, i.e. the maximal processing time appearing in the set of medium jobs is bounded by εOPT . On the other hand, the total area and the total processing time of the medium jobs should be small enough.

► **Lemma 3.** *Consider the sequence $\gamma_0 = \varepsilon$, $\gamma_{i+1} = \gamma_i \varepsilon^4$. There exists an $i \in \{1, \dots, 1/\varepsilon\}$ such that*

$$\frac{1}{m}p(\mathcal{J}_{\gamma_i}) + \frac{2}{R}\text{area}(\mathcal{J}_{\gamma_i}) \leq \varepsilon \left(\frac{1}{m}p(\mathcal{J}) + \frac{2}{R}\text{area}(\mathcal{J}) \right), \quad (1)$$

where $\mathcal{J}_{\gamma_i} := \{j \in \mathcal{J} \mid p(j) \in [\gamma_i T, \gamma_{i-1} T)\}$ and we can find this i in $\mathcal{O}(n + 1/\varepsilon)$.

Let $i \in \{1, \dots, 1/\varepsilon\}$ be the smallest value such that \mathcal{J}_{γ_i} has the property from Lemma 3 and define $\mu := \gamma_i$ and $\delta := \gamma_{i-1}$. Note that $\gamma_i = \varepsilon^{1+4i}$ and hence $\delta \geq \varepsilon^{4/\varepsilon+1}$. Using these values for δ and μ , we partition the set of jobs into large $\mathcal{J}_L := \{j \in \mathcal{J} | p(j) \geq \delta T\}$, small $\mathcal{J}_S := \{j \in \mathcal{J} | p(j) < \mu T\}$ and medium $\mathcal{J}_M := \{j \in \mathcal{J} | \mu T \leq p(j) < \delta T\}$.

► **Lemma 4.** *The medium jobs can be scheduled in $\mathcal{O}(n \log(n))$ operations with makespan $\mathcal{O}(\varepsilon)T$*

The final simplification step is to round the processing times of the large jobs using the rounding in Lemma 5 to multiples of $\varepsilon\delta T$.

► **Lemma 5** (See [23]). *Let $\delta \geq \varepsilon^k$ for some value $k \in \mathbb{N}$. At loss of a factor $(1 + 2\varepsilon)$ in the approximation ratio, we can round the processing time of each job j with processing time $\varepsilon^{l-1}T \geq p(j) \geq \varepsilon^l T$ for some $l \in \mathbb{N} \leq k$ such that it has a processing time $k_j \varepsilon^{l+1}T$ for a value $k_j \in \{1/\varepsilon, \dots, 1/\varepsilon^2 - 1\}$. Furthermore, the jobs can be started at a multiple of $\varepsilon^{l+1}T$.*

In this step, we reduce the number of different processing times of large jobs to $\mathcal{O}(\log_\varepsilon(\delta)/\varepsilon^2) = \mathcal{O}(1/\varepsilon^3)$. However, we lengthen the schedule at most by the factor $(1 + 2\varepsilon)$. Furthermore, this rounding reduces the starting times of these jobs to at most $\mathcal{O}(1/(\varepsilon\delta))$ possibilities since all the large jobs start and end at multiples of $\varepsilon\delta T$ and the optimal makespan of the rounded instance is bounded by $(1 + 2\varepsilon) \cdot 4T$.

Scheduling Large Jobs. In this section, we describe how to schedule the large jobs when given the size of the makespan $T' := l\varepsilon T$ of the rounded schedule. For a given set \mathcal{S} of start and endpoints of long jobs, we define a layer l_i as the processing time between two consecutive starting times $s_i, s_{i+1} \in \mathcal{S}$. Notice that, during the processing of a layer in a rounded optimal schedule, the resource requirement and number of machines used by large jobs stays unchanged since the large jobs only start and end at the starting points in \mathcal{S} .

► **Lemma 6.** *Let $\gamma \in (0, 1]$ and $T' = l\varepsilon T \geq \text{OPT}$ for some $l \in \mathbb{N}$, and $\bar{\mathcal{J}}$ be a set of jobs for which an optimal schedule exists such that all jobs in $\bar{\mathcal{J}}$ have their starting and endpoints in \mathcal{S} . There exists an algorithm that finds in $\mathcal{O}((m \log(R))^{\mathcal{O}_\varepsilon(1)|\mathcal{S}|/\gamma})$ operations $\mathcal{O}((m \log(R))^{\mathcal{O}_\varepsilon(1)/\gamma})$ schedules with the following properties*

1. *In each of the schedules, all large jobs are scheduled except for a set $\mathcal{J}' \subseteq \bar{\mathcal{J}}$ of at most $|\mathcal{J}'| \in 3|\mathcal{S}|$ jobs and a total resource requirement of at most $R(\mathcal{J}') \leq \gamma R$.*
2. *In at least one of the schedules, in each layer given by \mathcal{S} , the total number of machines and resources not used by jobs in $\bar{\mathcal{J}}$ is as large as in a rounded optimal schedule.*

In the APTAS we will use the algorithm from Lemma 6 with $\bar{\mathcal{J}} := \mathcal{J}_L$ and $\mathcal{S}' := \mathcal{S}$.

Scheduling Small Jobs. We will schedule small jobs inside the layers using the residual free resources and machines given by the guess for the large jobs. We define $m_s^{(S)}$ as the number of machines in layer s not used by jobs with processing times larger than δT and analogously define $R_s^{(S)}$ as the number of resources not used by jobs with processing times larger than δT during the processing of this layer in an optimal schedule.

► **Lemma 7.** *Define for each layer $s \in \mathcal{S}$ a box with processing time $(1 + \varepsilon)\varepsilon\delta T$, $m_s^{(S)}$ machines and $R_s^{(S)}$ resources, where the values $m_s^{(S)}$ and $R_s^{(S)}$ are at least as large as in a rounded optimal schedule. There exists an algorithm with time complexity $\mathcal{O}(n) \cdot \mathcal{O}_{\varepsilon, |\mathcal{S}|}(1)$, that places the jobs inside the boxes and an additional horizontal box with m machines, R resources, and processing time $\mathcal{O}(\varepsilon)T$.*

This algorithm uses the same techniques as the AFPTAS designed by Jansen et al. [21]. For the sake of completeness the ideas can be found in the appendix.

The AFPTAS. We can summarize the algorithm as follows. We define $T := \min\{p_{\max}, \text{area}(\mathcal{J})/R, p(\mathcal{J})/m\}$ and simplify the instance as described above. Then via a binary search framework, we try values $T' \in [T, 4T]$ as optimal makespan. For each of the considered values T' , we use the algorithm from Lemma 6 to generate several schedules for the large jobs each with makespan at most T' . If we cannot find a feasible schedule, the value T' was too small. Otherwise, we use the algorithm from Lemma 7 to schedule the small jobs inside each of the generated schedules for the large jobs. If T' was large enough, we can place the small jobs inside the layers, by increasing the schedule by a factor of at most $\mathcal{O}(\varepsilon)$. If the small jobs do not fit in one of the schedules for large jobs, the chosen T' was too small. If we have found a schedule for the small jobs, we try the next smaller value for T' in binary search fashion. In the final step, we use greedy list schedule to schedule the medium jobs and place the set \mathcal{J}' of non scheduled large jobs at the top.

3 A $(3/2 + \varepsilon)$ -Approximation

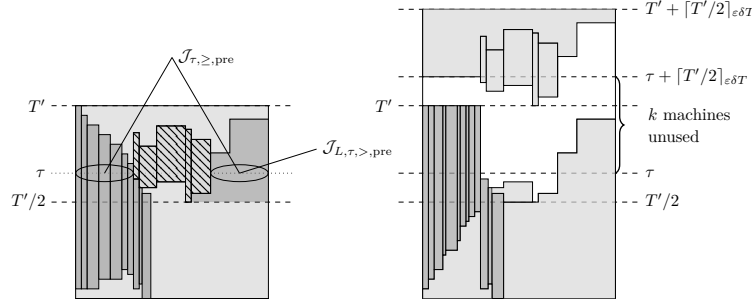
We aim to find a schedule with makespan $(3/2 + \mathcal{O}(\varepsilon))T'$, where T' is the assumed optimal makespan given by a binary search framework. Consider Lemma 6. If one of the jobs in \mathcal{J}' has a processing time larger than $T'/2 + \mathcal{O}(\varepsilon)T$, we exceed the aspired approximation ratio of $(3/2 + \mathcal{O}(\varepsilon))T$, when placing the set \mathcal{J}' on top of the schedule. We call this set of critical jobs huge jobs, i.e., $\mathcal{J}_H := \{j \in \mathcal{J} | p(j) > T'/2\}$, and redefine the set of large jobs as $\mathcal{J}_L := \{j \in \mathcal{J} | \delta T \leq p(j) \leq T'/2\}$ respectively.

Notice that the processing of all huge jobs has to intersect the time $T'/2$ in each schedule with makespan at most T' and each machine can contain at most one of these jobs. If we could guess the starting positions of these huge jobs, and schedule only the large jobs with the algorithm from Lemma 6, the discarded jobs \mathcal{J}' would have a processing time of at most $T'/2$ and could be placed on top of the schedule, resulting in a schedule of makespan at most $(3/2 + \mathcal{O}(\varepsilon))T'$. Sadly this guessing step is not possible in polynomial time since there are up to m of these jobs and iterating all combinations of their starting position needs $\Omega((1/\varepsilon\delta)^m)$ operations. Our idea is to let almost all the huge jobs end at a common point in time, e.g. T' , and thus avoid the guessing step. To solve the violation of the resource or machine condition, we shift up all the jobs which start after $\lceil T'/2 \rceil_{\varepsilon\delta T}$ by $\lceil T'/2 \rceil_{\varepsilon\delta T}$ such that they now start after T' , where we denote by $\lceil T'/2 \rceil_{\varepsilon\delta T}$ the integer multiple of $\varepsilon\delta T$ that is the first which has a size of at least $T'/2$.

While this shift fixes the start positions of the huge jobs, the large jobs are again placed with the techniques described in Section 2. Since Lemma 6 states that each of the generated schedules may not schedule a subset \mathcal{J}' of the large jobs, we need to find a gap in the shifted schedule where we can place them. In the following, we will consider optimal schedules and the possibilities to rearrange the jobs. Depending on this arrangement, we can find a gap of processing time $\lceil T'/2 \rceil_{\varepsilon\delta T}$ for the fractional scheduled large jobs.

Let us assume that we have to schedule $k := |\mathcal{J}'| \in \mathcal{O}_\varepsilon(1) \leq m/4$ jobs with total resource requirement at most $\gamma R \leq R/(3|\mathcal{S}|)$. We consider an optimal schedule, after applying the simplification steps and the corresponding transformed optimal schedule, where each large job starts at a multiple of $\varepsilon\delta T$ and each huge job starts at a multiple of $\varepsilon^2 T$. Furthermore, we will assume that there are more than $4k = \mathcal{O}_\varepsilon(1)$ huge jobs. Otherwise, we can guess their starting positions in $\mathcal{O}((1/\varepsilon\delta)^{4k})$ and place the fractional scheduled large jobs on top of the schedule.

In the following, we will prove that by extending it by $\lceil T'/2 \rceil_{\varepsilon\delta T}$, we can transform the rounded optimal schedule $\text{OPT}_{\text{rounded}}$ such that all the huge jobs, except for $\mathcal{O}(k)$ of them, end at a common point in time and we can place k further narrow large jobs without violating the machine or the resource constraint.



■ **Figure 1** In this and the following figures we present the processing time on the y-axis, while the resource requirements of jobs can be found on the x-axis. While the machines are not visually represented in these figures, the machine condition has to apply for each horizontal cut through the schedule. **On the left:** A rounded optimal schedule. The hatched rectangles are the jobs that start after $T'/2$ and intersect τ , the dark gray area corresponds to large jobs, which start before $T'/2$ and end after τ and the dark gray rectangles on the left are huge jobs. **On the right:** The corresponding shifted schedule.

► **Lemma 8.** Let $k := |\mathcal{J}'| \leq m/4$ and $\gamma \leq 1/(3|\mathcal{S}|)$. Furthermore, let a rounded optimal schedule $\text{OPT}_{\text{rounded}}$ with makespan at most T' and at most $|\mathcal{S}|$ starting positions for large jobs be given.

Without removing any job from the schedule, we can find a transformed schedule $\text{OPT}_{\text{shift}}$ with makespan at most $T' + \lceil T'/2 \rceil_{\epsilon\delta T}$, with the following properties:

1. We can guess the end positions of all huge jobs in polynomial time.
2. There is a gap of processing time $\lceil T'/2 \rceil_{\epsilon\delta T}$ with k empty machines and γR free resources where we can schedule the jobs in \mathcal{J}' .
3. There is an injection which maps each layer s in $\text{OPT}_{\text{rounded}}$ with $m_{s,S}$ machines and $R_{s,S}$ resources not used by huge and large jobs to a layer in $\text{OPT}_{\text{shift}}$ where there are at least as many machines and resources not used by these jobs.

Proof. We will prove this lemma by a careful analysis of the structure of the schedule $\text{OPT}_{\text{rounded}}$. First, however, we introduce some notations. Let $s \in \mathcal{S}$, with $s > T'/2$ be any starting point of large jobs. We say a job $j \in \mathcal{J}$ intersects s or is intersected by s if $\sigma(j) < s < \sigma(j) + p(j)$. We will differentiate sets of jobs that start before $T'/2$ and those that start at or after $T'/2$ by adding the attribute pre to sets of jobs that contain only jobs starting before $T'/2$, and the attribute post to those that contain only jobs that start at or after $T'/2$. Furthermore, we will identify the sets of jobs that intersect certain points of time. We will add the attribute s_{\geq} to denote a set of jobs that is processed at least until the point in time $s \in \mathcal{S}$, i.e., we denote by $\mathcal{J}_{s_{\geq}, \text{pre}} := \{j \in \mathcal{J} | p(j) \geq \delta T, \sigma(j) < T'/2, \sigma(j) + p(j) \geq s\}$ the set of large and huge jobs starting before $T'/2$ and ending at or after s . On the other hand, if we are only interested in the jobs that intersect the time s , we add the attribute $s_{>}$ to the set and mean $\mathcal{J}_{s_{>}, \text{pre}} := \{j \in \mathcal{J} | p(j) \geq \delta T, \sigma(j) < T'/2, \sigma(j) + p(j) > s\}$. Finally, we will indicate if the set contains only huge or only large jobs, by adding the attribute H or L .

Let $\tau \in \{s | s \in \mathcal{S}, T'/2 \leq s \leq T'\}$ be the smallest value such that there are at most $m - k$ jobs (huge or large) that start before $T'/2$ and intersect τ , i.e., end after τ . We partition the set of large jobs intersected by τ into two sets. Let $\mathcal{J}_{L, \tau, >, \text{pre}} := \{j \in \mathcal{J}_L | \sigma(j) < T'/2, \sigma(j) + p(j) > \tau\}$ be the set of large jobs which start before $T'/2$ and end at or after τ . Further let $\mathcal{J}_{L, \tau, >, \text{post}} := \{j \in \mathcal{J}_L | T'/2 \leq \sigma(j) < \tau, \sigma(j) + p(j) > \tau\}$ be the set of large jobs, which are started at or after $T'/2$ but before τ and end after τ , see Figure 1.

Note that by the choice of τ in each point between τ and $T'/2$ in the schedule there are more than $m - k$ machines used by $\mathcal{J}_{\tau, \geq, \text{pre}}$. As a result there are at most $k - 1$ machines used by jobs starting after $T'/2$ at each point between $T'/2$ and τ , implying $|\mathcal{J}_{L, \tau, >, \text{post}}| < k$.

We now construct a shifted schedule. Starting times in this schedule will be denoted by σ' . We shift each job $j \in \mathcal{J}$ with $\sigma(j) \geq T'/2$ and $\sigma(j) + p_j \geq \tau$ exactly $\lceil T'/2 \rceil_{\varepsilon \delta T}$ upwards, i.e., we define $\sigma'(j) := \sigma(j) + \lceil T'/2 \rceil_{\varepsilon \delta T}$ for these jobs. Furthermore, each huge job $j \in \mathcal{J}_H$ intersecting τ is shifted upwards such that it ends at T' , i.e., we define $\sigma'(j) := T' - p_j$ for these jobs j , see Figure 1. Note that there are at most k huge jobs ending strictly before τ . If the total number of huge jobs ending before or at τ is larger than k , we choose arbitrarily from the set of jobs ending at τ and shift them until there are exactly k huge jobs ending before or at τ .

▷ **Claim 9.** After this shift there are at least k machines at each point between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon \delta T}$ that are not used by any other job.

Proof. Up to T' , there are k free machines, because there is no new job starting between τ and T' since we shifted all of them up such that they start after $\lceil T'/2 \rceil_{\varepsilon \delta T}$. On the other hand, only jobs from the set $\mathcal{J}_{L, \tau, >, \text{post}}$ are processed between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon \delta T}$. Since $|\mathcal{J}_{L, \tau, >, \text{post}}| < k$ and $m - k \geq k$ this leaves k free machines which proves the claim. ◁

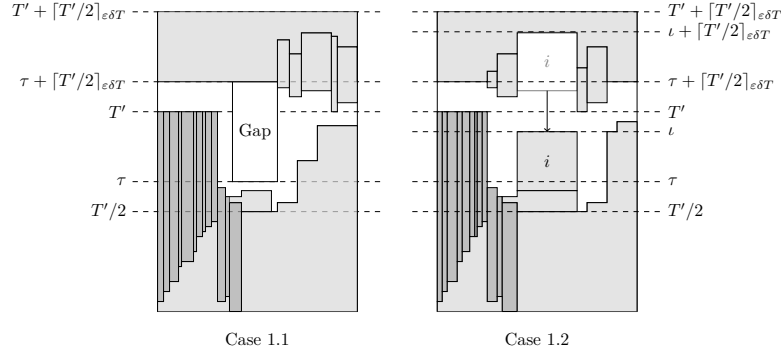
The idea is to place the gap at τ since there are enough free machines. However, it can happen that at a point between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon \delta T}$ there is not enough free resource for the gap. In the following, we carefully analyze where we can place the k jobs, dependent on the structure of the optimal schedule $\text{OPT}_{\text{rounded}}$

Case 1: $r(\mathcal{J}_{\tau, \geq, \text{pre}}) \leq R - \gamma R$. In this case there are at least γR free resources at each point in the shifted schedule between τ and T' since there are no jobs starting between these points of time. To place the k fractional scheduled jobs, we have to generate a gap of processing time $\lceil T'/2 \rceil_{\varepsilon \delta T}$. In this gap there have to be k unused machines and γR unused resources. For the time between τ and T' , we have this guarantee, while for the time between T' and $\tau + \lceil T'/2 \rceil_{\varepsilon \delta T}$, we have k free machines, but might have less than γR free resource. The only jobs overlapping in this time window are the jobs from the set $\mathcal{J}_{L, \tau, >, \text{post}}$, see Figure 1. If these jobs have a small resource requirement, we have found our gap, see Case 1.1. and, otherwise, we have to look more careful at the schedule.

Case 1.1: $r(\mathcal{J}_{L, \tau, >, \text{post}}) \leq R - \gamma R$. In this case, the required gap is positioned between τ and $\tau + \lceil T'/2 \rceil_{\varepsilon \delta T}$, see Figure 2. In this shifted optimal schedule there are at most k huge jobs ending before τ . In the algorithm, we will guess τ dependent on a given solution for the large jobs and guess these k huge jobs and their start points in $\mathcal{O}(m^k S^k)$, which is polynomial in the input size, see Section 3 for an overview.

Case 1.2: $r(\mathcal{J}_{L, \tau, >, \text{post}}) > R - \gamma R$. In this case, there is a point $t \in [\tau, T']$ such that after this point there are less than γR free resources. Therefore, we need another position to place the fractionally scheduled jobs. We partition the set $\mathcal{J}_{L, \tau, >, \text{post}}$ into at most $|\mathcal{S}|/2$ sets $\mathcal{J}_{L, \tau, >, \text{post}}^\iota$ by their original finishing points $\iota \in \mathcal{S}_{>\tau}$, i.e., each job in $\mathcal{J}_{L, \tau, >, \text{post}}^\iota$ finishes at ι in the non shifted rounded optimal schedule.

▷ **Claim 10.** One of the sets $\mathcal{J}_{L, \tau, >, \text{post}}^\iota$, $\iota \in \mathcal{S}_{>\tau}$, has a resource requirement of at least γR .



■ **Figure 2** Examples for the two Cases 1.1. and 1.2. In Case 1.1 the gap is positioned between τ and $\tau + \lceil T'/2 \rceil_{\epsilon \delta T}$. In Case 1.2 the jobs in the set $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$ are shifted back down. At each point between ι and $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$ there are at least γR unused resources.

Proof. The jobs in $\mathcal{J}_{L,\tau,>,\text{post}}$ use more than $R - \gamma R$ resource in total. Since $\gamma \leq 1/(3|\mathcal{S}|) \leq 1/(|\mathcal{S}|/2 - 1)$, it holds that

$$\frac{R - \gamma R}{|\mathcal{S}|/2} \geq \frac{(1 - 1/(|\mathcal{S}|/2 - 1))R}{|\mathcal{S}|/2} = R/(|\mathcal{S}|/2 - 1) \geq \gamma R.$$

Hence, by the pigeon principle, one of the sets, say $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$, must have a resource requirement of at least γR . \triangleleft

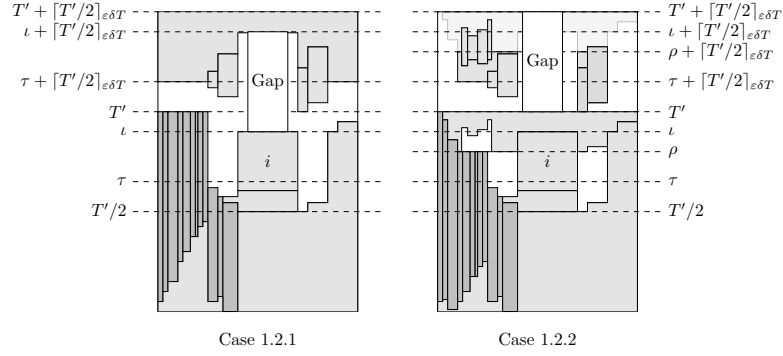
Let $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$ be this set. To generate a gap, we shift down all jobs in $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$ back to their primary start position, see Figure 2.

▷ **Claim 11.** As a result of this shift, there are at least γR free resources at each point between ι and $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$.

Proof. At each point between ι and T' there were γR unused resources before. Each job which starts between T' and $\tau + \lceil T'/2 \rceil_{\epsilon \delta T}$ is an element of $\mathcal{J}_{L,\tau,>,\text{post}}$ and was therefore scheduled in parallel to the jobs in $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$. Therefore, at each point between T' and $\tau + \lceil T'/2 \rceil_{\epsilon \delta T}$ at least γR resources are unused. From $\tau + \lceil T'/2 \rceil_{\epsilon \delta T}$ to $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$ the jobs $\mathcal{J}_{L,\tau,>,\text{post}}^\iota$ were scheduled, so there are at least γR free resources. \triangleleft

We now have to differentiate if there are at least k machines unused between $\tau + \lceil T'/2 \rceil_{\epsilon \delta T}$ and $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$, see Figure 2. Let $\rho \in \{s | \tau \leq s \leq T, s \in \mathcal{S}\}$ be the first point in the schedule where at most k jobs from $\mathcal{J}_{\tau,\geq,\text{pre}}$ are scheduled in the given optimal schedule (not the shifted one), i.e., ρ is the first point in time where $|\mathcal{J}_{\rho,\geq,\text{pre}}| \leq k$. Note that as a consequence $|\mathcal{J}_{\rho,\geq,\text{pre}}| \geq k$ since otherwise there would have been a point in time before ρ , where at most k machines are used by jobs starting before $T'/2$. We know that between T' and $\rho + \lceil T'/2 \rceil_{\epsilon \delta T}$ there always will be k machines unused since before the first shift they were blocked by jobs in $\mathcal{J}_{\tau,\geq,\text{pre}}$.

Case 1.2.1: $\rho \geq \iota$. In this case, at each point between ι and $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$ there are k machines unused. Between ι and T' there are k free machines by the choice of τ and between T' and $\rho + \lceil T'/2 \rceil_{\epsilon \delta T}$ there are k free machines by the choice of ρ . Therefore, there is a gap between ι and $\iota + \lceil T'/2 \rceil_{\epsilon \delta T}$, see Figure 3. Similar as in Case 1.1. the total number of guesses needed to place the huge jobs is bounded by $m_\epsilon^\mathcal{O}(1)$, although we have to add the guess for ρ .



■ **Figure 3** Examples for the shifted schedule and the position of the gap in the Cases 1.2.1 and 1.2.2.

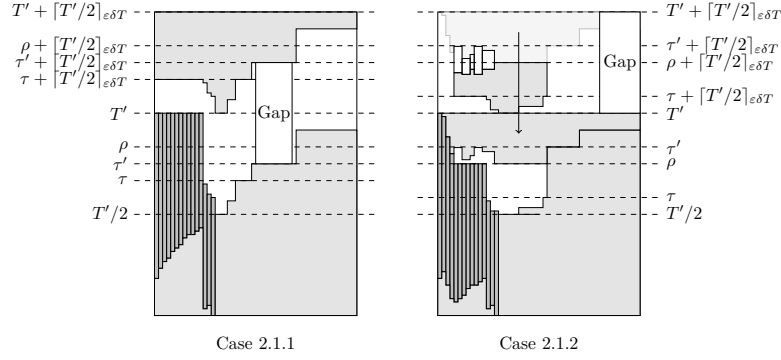
Case 1.2.2: $\rho < \iota$. Let $\mathcal{J}_{H,\rho} := \{j \in \mathcal{J}_H | s_j + p_j > \rho\}$ be the set of huge jobs, which are still scheduled after ρ . It holds that $|\mathcal{J}_{H,\rho}| \leq k$. As a consequence, it is possible to guess their starting positions in polynomial time. Therefore, the algorithm will schedule each job in $\mathcal{J}_{H,\rho}$ as in the original simplified schedule $\text{OPT}_{\text{rounded}}$. The other huge jobs, which end between τ and ρ , are scheduled such that they end at ρ , i.e., we define $\sigma'(j) := \rho - p(j)$ for each of these huge jobs j . Next, we shift the all the jobs j with starting time $\sigma'(j) \geq \rho + [T'/2]_{\epsilon\delta T}$ downwards such that they start as they had started before the first shift. As a result between T' and $T' + [T'/2]_{\epsilon\delta T}$, there are just jobs left which overlap the time from $\tau + [T'/2]_{\epsilon\delta T}$ to $\rho + [T'/2]_{\epsilon\delta T}$, see Figure 3. By the choice of ρ and τ at each point between $\tau + [T'/2]_{\epsilon\delta T}$ and $\rho + [T'/2]_{\epsilon\delta T}$ there are at most $m - k$ jobs which use at most $R - \gamma R$ resource since the job i was scheduled there before. Since each job between T' and $T' + [T'/2]_{\epsilon\delta T}$ overlaps this area there are at least k free machines and γR free resources in this area. Hence, we position the gap at T' . In the algorithm, we will guess τ and ρ dependent on a given fractional solution for the large jobs and guess the at most k jobs ending before τ and the k jobs ending after ρ in $\mathcal{O}(m^{2k-1})$. For each of these jobs, we have to guess its starting time out of at most $|\mathcal{S}|/2$ possibilities.

Case 2: $r(\mathcal{J}_{\tau,\geq,\text{pre}}) > R - \gamma R$. In this case, the gap has to start strictly after τ since at τ there is not enough free resource. Let $\mathcal{J}_{L,T'/2}$ be the set of large jobs intersecting the point in time $T'/2$. Remember that $\mathcal{J}_{\tau,\geq,\text{pre}}$ contains huge and large jobs. Since $r(\mathcal{J}_{\tau,\geq,\text{pre}}) > R - \gamma R$ at least one of these sets of jobs (huge or large) has to contribute a large resource requirement to $r(\mathcal{J}_{\tau,\geq,\text{pre}})$. In the following, we will find the gap, depending on which of both sets contributes a suitable large resource requirement.

Case 2.1: $r(\mathcal{J}_{L,T'/2}) \geq 2\gamma R$. Let $\tau' \in \{s \in \mathcal{S} | \tau \leq s \leq T'\}$ be the first point in time where $r(\mathcal{J}_{L,T'/2}) - r(\mathcal{J}_{L,\tau',>,\text{pre}}) \geq \gamma R$. Note that $\tau \leq \tau'$ since, otherwise, there would be γR free resources at τ .

▷ **Claim 12.** By this choice at each point between τ' and $\tau' + [T'/2]_{\epsilon\delta T}$ there are at least γR free resources.

Between τ' and T' there are γR free resources since jobs from $\mathcal{J}_{L,T'/2}$ with a resource requirement of at least γR end before τ' . On the other hand, before the shift there was at least γR resource blocked by jobs from $\mathcal{J}_{L,T'/2}$ between $T'/2$ and τ' and hence after the shift there is at least γR free resource at any time between T' and $\tau' + [T'/2]_{\epsilon\delta T}$. Moreover, as in Case 1.2, let $\rho \in \{s | \tau \leq s \leq T', s \in \mathcal{S}\}$ be the first point in the schedule where $|\mathcal{J}_{\rho,>,\text{pre}}| \leq k$, i.e., where at most k jobs are scheduled that start before $T'/2$.



■ **Figure 4** Examples for the shifted schedules and the position of the gap in Cases 2.1.1 and Case 2.1.2.

▷ **Claim 13.** By this choice at each point between τ and $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$ there are at least k unused machines.

From τ to T' there are k unused machines, by the choice of τ . On the other hand, at each point in time between T' and $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$ there where k machines blocked by jobs from that started before $T'/2$ and these machines are now unused.

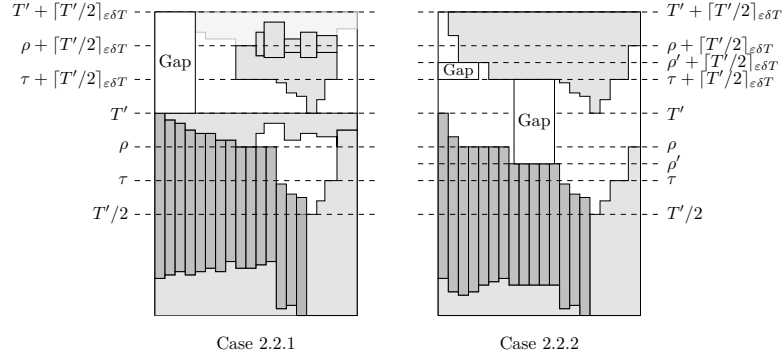
Similar as in Cases 1.2.1 and 1.2.2, we will find the gap dependent of the relation between τ' and ρ .

Case 2.1.1: $\rho \geq \tau'$. In this case between τ' and $\tau' + \lceil T'/2 \rceil_{\epsilon\delta T}$ there are at least k unused machines. Therefore, we have a gap between these two points, which is large enough, see Figure 4. In the algorithm, we have to guess the k huge jobs, which end before τ and their start point, as well as the points τ , τ' and ρ . All the guesses for this case can be iterated in polynomial time.

Case 2.1.1: $\rho < \tau'$. In this case, we act like in Case 1.2.2 and shift all huge jobs, but the at most k jobs ending after ρ , downwards such that they end at ρ , see Figure 4. Furthermore, we shift all jobs starting after $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$ back downwards such that they again start at their primary start position. Now after T' there are just jobs having their start or end position between $\tau + \lceil T'/2 \rceil_{\epsilon\delta T}$ and $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$. At each point between these two points there are at least k unused machines and γR unused resource with the same arguments as in Case 1.2.2. Hence, we have a gap with the right properties between T' and $T' + \lceil T'/2 \rceil_{\epsilon\delta T}$. All the guesses for this case can be iterated in polynomial time.

Case 2.2: $r(\mathcal{J}_{L, T'/2}) < 2\gamma R$. Since we have $r(\mathcal{J}_{\tau, \geq, \text{pre}}) > R - \gamma R$ (by Case 2.) and it holds that $(\mathcal{J}_H \cap \mathcal{J}_{\tau, \geq, \text{pre}}) \cup (\mathcal{J}_{L, T'/2} \cap \mathcal{J}_{\tau, \geq, \text{pre}}) = \mathcal{J}_{\tau, \geq, \text{pre}}$ we get that $r(\mathcal{J}_H \cap \mathcal{J}_{\tau, \geq, \text{pre}}) \geq R - 3\gamma R$. Similar as before, let $\rho \in \{s | \tau \leq s \leq T, s \in S\}$ be the first point in the schedule where less than k jobs are scheduled that start before $T'/2$. By the same argument as in Case 2.1, we know that at every point between τ and $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$ there are at least k unused machines in the shifted schedule.

Case 2.2.1: $r(\mathcal{J}_{\rho, \geq, \text{pre}}) \geq \gamma R$. In this case, we can construct a schedule in the same way as in case 1.2.2 or 2.1.2 by shifting down the jobs that start after $\rho + \lceil T'/2 \rceil_{\epsilon\delta T}$ and positioning the gap at T' , see Figure 5. This is possible because the jobs that are scheduled



■ **Figure 5** The shifted schedule and the position of the gap in Cases 2.2.1 and 2.2.2. Note that in Case 2.2.2 the gap is not displayed continuously. However, by swapping the used resource, we can make it continuous. We only need the fact that at each point in time there are enough free resources and machines.

between $\tau + \lceil T'/2 \rceil_{\varepsilon\delta T}$ and $\rho + \lceil T'/2 \rceil_{\varepsilon\delta T}$ can use at most $R - \gamma R$ resources in this case since $r(\mathcal{J}_{\rho, \geq, \text{pre}}) \geq \gamma R$ and hence at least γR resources are blocked by the jobs in $\mathcal{J}_{\rho, \geq, \text{pre}}$. All the guesses for this case can be iterated in polynomial time.

Case 2.2.2: $r(\mathcal{J}_{\rho, \geq, \text{pre}}) < \gamma R$. Let $\rho' \in \{i\delta^2 | \tau/\delta^2 \leq i \leq \rho/\delta^2, i \in \mathbb{N}\}$ be the smallest value, where $r(\mathcal{J}_{\rho', \geq, \text{pre}}) \leq \gamma R$. Remember, we had $r(\mathcal{J}_H \cap \mathcal{J}_{\tau, \geq, \text{pre}}) \geq R - 3\gamma R$ so huge jobs with summed resource requirement of at least $R - 4\gamma R$ are finished till ρ' . We partition the huge jobs that finish between τ and ρ by their processing time. Since each job has a processing time of at least $\lceil T'/2 \rceil_{\varepsilon\delta T}$, we get at most $\mathcal{O}(1/\varepsilon\delta) \leq |\mathcal{S}|/2$ sets. As seen in Section 2, we have to discard at most $k \leq 3|\mathcal{S}|$ large jobs, which have to be placed later on.

▷ **Claim 14.** There exists a set in the partition, which uses at least $3\gamma R$ resource total.

Proof. Since $\gamma \leq 1/(2|\mathcal{S}|) \leq 1/(3|\mathcal{S}|/2 + 4)$ it holds that

$$\frac{R - 4\gamma R}{|\mathcal{S}|/2} \geq \frac{(1 - 4/(3|\mathcal{S}|/2 + 4))R}{|\mathcal{S}|/2} = 3R/(3|\mathcal{S}|/2 + 4) \geq 3\gamma R.$$

Therefore, by the pigeon principle, there must be one set in the partition, which has summed resource requirement of at least $3\gamma R$. ◁

We sort the jobs in this partition by non increasing order of resource requirement. We greedily take jobs from this set, till they have a summed resource requirement of at least γR and schedule them such that they end before ρ' . If there was a job with more than γR resource requirement, it had to be finished before ρ' since the resource requirement of huge jobs finishing after ρ' is smaller than γR and we only chose it. Otherwise, the greedily chosen jobs have summed resource requirement of at least $2\gamma R$. Since the considered set has a summed resource requirement of at least $3\gamma R$, jobs of this set with summed resource requirement at least $2\gamma R$ end before ρ' . Therefore, we do not violate any constraint by shifting down these jobs such that they end at ρ' , see Figure 5.

Concerning property three, note that since we only use the free area (machines and resources) to schedule the k large jobs inside the gap, there is a layer s' in the shifted schedule, for each layer $s \in \mathcal{S}$ that has at least as many machines and resources not used by large and huge jobs as the layer s . ◀

Algorithm Summary. Given a value $T' := i\varepsilon'T$, we determine the set \mathcal{S} and call the algorithm from Lemma 6 with $\gamma = 1/(3|\mathcal{S}| + 4)$ to generate the set of schedules for the large jobs. One of these schedules uses in each layer at most as many machines and resources for large jobs, as the rounded optimal schedule, or the value T' is too small. Furthermore, the set of not scheduled large jobs \mathcal{J}' has a total machine requirement of at most $3|\mathcal{S}|$, a total resource requirement of at most γR , and each job has a processing time of at most $T'/2$.

For each of these schedules, the algorithm iterates all values for τ and ρ and all possibilities for the at most $2k$ huge jobs ending before or after these values and their starting positions. Then, we identify the case and the other variables dependent on the guesses and the solution schedule for the large jobs. By this we generate a new set of schedules, for which it will try to place the small jobs. We refer to the full version for more details.

To bound the total number of guesses that we add by this procedure, note that we have to guess τ , ρ and ρ' from at most $\mathcal{O}(|\mathcal{S}|)$ possibilities. Further, we for each of these guesses, the algorithm guesses at most $2k$ huge jobs and their starting positions. The total number of these guesses is bounded by $(m/\varepsilon^2)^{\mathcal{O}(k)}$, since the huge jobs start at multiples of $\varepsilon^2 T$. Therefore, the total number of guesses for the large jobs is bounded by $(m/\varepsilon^2)^{\mathcal{O}(k)} \cdot \mathcal{O}(|\mathcal{S}|^3)$. Since $k \leq 3|\mathcal{S}|$, this guess for the huge jobs lengthens the running time by a factor of at most $(m/\varepsilon)^{1/\varepsilon^{\mathcal{O}(1/\varepsilon^2)}}$. This concludes the proof of Theorem 1.


References

- 1 Anna Adamaszek, Tomasz Kociumaka, Marcin Pilipczuk, and Michał Pilipczuk. Hardness of approximation for strip packing. *TOCT*, 9(3):14:1–14:7, 2017. doi:10.1145/3092026.
- 2 Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
- 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 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.
- 6 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.
- 7 Jianzhong Du and Joseph Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989. doi:10.1137/0402042.
- 8 Leah Epstein and Asaf Levin. AFPTAS results for common variants of bin packing: A new method for handling the small items. *SIAM Journal on Optimization*, 20(6):3121–3145, 2010. doi:10.1137/090767613.
- 9 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.
- 10 Michael R. Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975. doi:10.1137/0204015.
- 11 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.
- 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 Ronald L Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics* 17.2, pages 416–429, 1969.
- 14 Alexander Grigoriev, Maxim Sviridenko, and Marc Uetz. Machine scheduling with resource dependent processing times. *Mathematical programming*, 110(1):209–228, 2007.
- 15 Rolf Harren, Klaus Jansen, Lars Prädel, 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. *Theory of Computing Systems*, 2019. doi:10.1007/s00224-019-09910-6.
- 18 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.
- 19 Klaus Jansen. A $(3/2 + \epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 224–235, 2012. doi:10.1145/2312005.2312048.
- 20 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.
- 21 Klaus Jansen, Marten Maack, and Malin Rau. Approximation schemes for machine scheduling with resource (in-)dependent processing times. *ACM Trans. Algorithms*, 15(3):31:1–31:28, 2019. doi:10.1145/3302250.
- 22 Klaus Jansen and Lorant Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002. doi:10.1007/s00453-001-0085-8.
- 23 Klaus Jansen and Malin Rau. Improved approximation for two dimensional strip packing with polynomial bounded width. *Theor. Comput. Sci.*, 789:34–49, 2019. doi:10.1016/j.tcs.2019.04.002.
- 24 Klaus Jansen and Malin Rau. Linear time algorithms for multiple cluster scheduling and multiple strip packing. *CoRR*, abs/1902.03428, 2019. arXiv:1902.03428.
- 25 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.
- 26 Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, 39(8):3571–3615, 2010. doi:10.1137/080736491.
- 27 Berit Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006. doi:10.1007/s10951-006-8497-6.
- 28 Hans Kellerer. An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. *Operations Research Letters*, 36(2):157–159, 2008.
- 29 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.
- 30 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.
- 31 Martin Niemeier and Andreas Wiese. Scheduling with an orthogonal resource constraint. *Algorithmica*, 71(4):837–858, 2015. doi:10.1007/s00453-013-9829-5.

- 32 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.
- 33 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.
- 34 A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997. doi:10.1137/S0097539793255801.
- 35 Maxim Sviridenko. A note on the kenyon-remila strip-packing algorithm. *Inf. Process. Lett.*, 112(1-2):10–12, 2012. doi:10.1016/j.ipl.2011.10.003.
- 36 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.

Certified Approximation Algorithms for the Fermat Point and n -Ellipses

Kolja Junginger 



Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Ioannis Mantas  

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Evanthia Papadopoulou  

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Martin Suderland  

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Chee Yap  

Courant Institute, New York University, NY, USA

Abstract

Given a set A of n points in \mathbb{R}^d with weight function $w : A \rightarrow \mathbb{R}_{>0}$, the Fermat distance function is $\varphi(\mathbf{x}) := \sum_{\mathbf{a} \in A} w(\mathbf{a}) \|\mathbf{x} - \mathbf{a}\|$. A classic problem in facility location dating back to 1643, is to find the *Fermat point* \mathbf{x}^* , the point that minimizes the function φ . We consider the problem of computing a point $\tilde{\mathbf{x}}^*$ that is an ε -approximation of \mathbf{x}^* in the sense that $\|\tilde{\mathbf{x}}^* - \mathbf{x}^*\| < \varepsilon$. The algorithmic literature has so far used a different notion based on ε -approximation of the value $\varphi(\mathbf{x}^*)$. We devise a certified subdivision algorithm for computing $\tilde{\mathbf{x}}^*$, enhanced by Newton operator techniques. We also revisit the classic Weiszfeld-Kuhn iteration scheme for \mathbf{x}^* , turning it into an ε -approximate Fermat point algorithm. Our second problem is the certified construction of ε -isotopic approximations of n -ellipses. These are the level sets $\varphi^{-1}(r)$ for $r > \varphi(\mathbf{x}^*)$ and $d = 2$. Finally, all our planar ($d = 2$) algorithms are implemented in order to experimentally evaluate them, using both synthetic as well as real world datasets. These experiments show the practicality of our techniques.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Fermat point, n -ellipse, subdivision, approximation, certified, algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.54

Funding The work of C. Yap was supported by an USI Visiting Professor Fellowship (Fall 2018); additional support came from NSF Grants # CCF-1564132 and # CCF-2008768. The other authors were partially supported by the SNF project 200021E_154387.

1 Introduction

A classic problem in Facility Location, see e.g., [21, 43], is the placement of a facility to serve a given set of demand points or customers so that the total transportation costs are minimized. The total cost at any point is interpreted as the sum of the distances to the demand points. The point that minimizes this sum is called the *Fermat Point*; see Figure 1. This is an old geometric problem that has inspired scientists over the last three centuries.

A *weighted foci set* is a non-empty finite set of (demand) points $A = \{\mathbf{a}_1, \dots, \mathbf{a}_n\}$ in \mathbb{R}^d associated with a positive weight function $w : A \rightarrow \mathbb{R}_{>0}$. Each $\mathbf{a} \in A$ is called a *focus* with weight $w(\mathbf{a})$. Let $W := \sum_{\mathbf{a} \in A} w(\mathbf{a})$. The *Fermat distance function* of A is given by

$$\varphi(\mathbf{x}) := \sum_{\mathbf{a} \in A} w(\mathbf{a}) \|\mathbf{x} - \mathbf{a}\|,$$



© Kolja Junginger, Ioannis Mantas, Evanthia Papadopoulou, Martin Suderland, and Chee Yap; licensed under Creative Commons License CC-BY 4.0

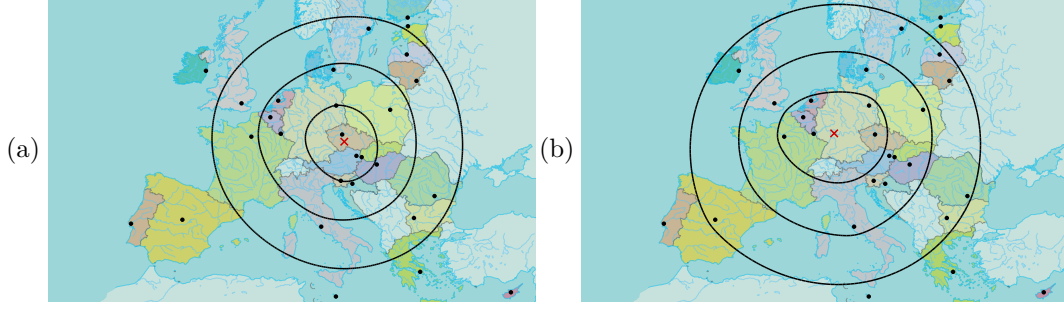
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 54; pp. 54:1–54:19

Leibniz International Proceedings in Informatics



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



■ **Figure 1** The Fermat point of the 28 EU-capitals (pre-Brexit), highlighted with (x), along with three 28-ellipses of different radii. (a) The foci (capitals) are unweighted. (b) Each focus has the weight of the country's population. The source of the map is <https://www.consilium.europa.eu>.

where $\|\cdot\|$ is the Euclidean norm in \mathbb{R}^d . The global minimum value of φ is called the *Fermat radius* of A , denoted by $r^* = r^*(A)$. Any point $\mathbf{x} \in \mathbb{R}^d$ that achieves this minimum, $\varphi(\mathbf{x}) = r^*$, is called a *Fermat point*, denoted by $\mathbf{x}^* = \mathbf{x}^*(A)$. The Fermat point is not unique if and only if A is collinear and n is even. We can check if A is collinear in $O(n)$ time, and in that case, the median, which is a Fermat point, can be found in $O(n \log n)$ time. So henceforth, we assume that A is not collinear. In that case φ is a strictly convex function [35, 37], and \mathbf{x}^* is unique.

We also consider the closely related problem of computing n -ellipses of A . For any $r > r^*(A)$, the level set of the Fermat distance function is $\varphi^{-1}(r) := \{\mathbf{x} \in \mathbb{R}^d : \varphi(\mathbf{x}) = r\}$. If $n = 1$, the level set is a sphere; and if $n = 2$ and $d = 2$, it is the classic ellipse. When A has n points, we call $\varphi^{-1}(r)$ an n -ellipsoid, or an n -ellipse if $d = 2$; hence the term *foci set*. From an application perspective, an n -ellipse of radius r can be viewed as a curve that bounds the candidate area for facility location [46], such that the total transportation cost to the demand points is at most r , as in Figure 1.

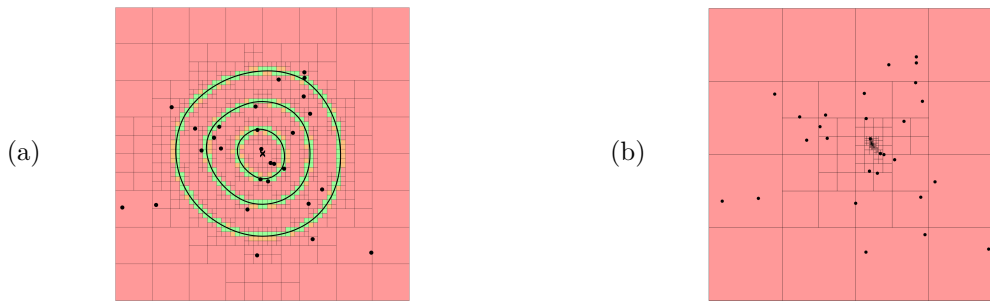
The question of approximating the Fermat point is of great interest as its coordinates are roots of polynomials of degree exponential in n [3]. For any $\varepsilon > 0$, an ε -approximation $\tilde{\mathbf{x}}^*$ to the Fermat point \mathbf{x}^* can be interpreted in 3 ways:

- (A) **Approximate Fermat Point:** $\|\tilde{\mathbf{x}}^* - \mathbf{x}^*\| \leq \varepsilon$;
- (B) **Absolute Approximate Fermat Radius:** $\varphi(\tilde{\mathbf{x}}^*) \leq \varphi(\mathbf{x}^*) + \varepsilon$;
- (C) **Relative Approximate Fermat Radius:** $\varphi(\tilde{\mathbf{x}}^*) \leq (1 + \varepsilon)\varphi(\mathbf{x}^*)$.

Thus, we have three ε -approximation problems: (A), (B) and (C). Essentially (B) and (C) are approximations of the Fermat radius, while (A) is a direct approximation of the Fermat point. In this paper we consider approximations in the sense of (A); to the best of our knowledge, only approximations in the sense of (B) and (C), have been considered before, see e.g., [8, 16]. Below, we show that (B) and (C) are easily *reduced* to (A) while the converse reductions are non-obvious (reductions in the sense of complexity theory).

In this work we introduce certified algorithms for approximating the Fermat point and n -ellipses, combining a subdivision approach with interval methods (cf. [33, 48]). The approach can be formalized in the framework of *soft predicates* [56]. Our certified algorithms are fairly easy to implement, and are shown to have good performance experimentally.

Related Work. The problem we study has a long history, with numerous extensions and variations. Out of the 15 names found in the literature, see [23], we call it *the Fermat point problem*. Other common names are the *Fermat-Weber problem* and the *Geometric median*.



■ **Figure 2** The resulting box subdivision of Figure 1(a) for (a) the n -ellipses and (b) the Fermat point.

problem. Apart from the Facility Location application introduced by Weber [57], the problem is motivated by applications in diverse fields such as statistics and data mining where it is known as the *1-Median problem*, and is an instance of the k -median clustering technique [27].

For $d = 2, n = 3$, the problem was first stated by P. Fermat (1607 - 1665) and was solved by E. Torricelli (1608 - 1647) and Krarup and Vajda [30] using a geometric construction. For $n = 4$, solutions were given by Fagnano [20] and Cieslik [14]. The first general method, for arbitrary n , is an iterative scheme proposed by Weiszfeld [58] in 1937. It was later corrected and improved by Kuhn [32] and Ostresh [43]; see Beck and Sabach [4] for a review. The method is essentially a gradient descent iterative algorithm. It behaves quite well in practice and has only linear convergence, with guaranteed convergence from any starting point.

A plethora of approximation algorithms for the Fermat point, in the senses of (B) and (C), can be found in the literature using various methods. There are algorithms based on semidefinite programming [45], interior point methods [16, 60], sampling [2, 16], geometric data structures [8] and coresets [26], among others [13, 22]. Moreover, special configurations of foci have been considered [7, 15], a continuous version of the problem [21], and also a generalized Fermat point of planar convex objects [1, 12, 18].

The literature on n -ellipses is smaller but equally old: Nagy [38] proved that n -ellipses are convex curves, calling them *egg curves*, and dating them back to Tschirnhaus in 1695 [55, p. 183]. Further, he characterized the singular points of the n -ellipses as being either foci or the Fermat point. Another early work is by Sturm in 1884 [53]. Sekino [51] showed that the Fermat distance function φ is C^∞ on $\mathbb{R}^2 \setminus A$. So, the n -ellipse is a piecewise smooth curve, as it may pass through several foci. Nie et al. [42] showed that the polynomial equation defining the n -ellipses has algebraic degree exponential in n .

Our Contributions. In this paper, we design, implement and experimentally evaluate algorithms for approximating the Fermat point of a given set of foci in \mathbb{R}^d . We also compute an ε -approximate n -ellipse; a problem not considered in computational literature before. These are the first certified algorithms [36, 54] for these problems. Our contributions are summarized as follows:

- We design two certified algorithm for the approximate Fermat point: one based on subdivision, the other based on Weiszfeld iteration [58].
- Our notion of ε -approximate Fermat point appears to be new; in contrast, several recent works focus on ε -approximation of the Fermat radius. The approximate Fermat radius can be reduced to approximate Fermat point; the converse reduction is unclear.
- Based on the *PV construction* [47, 33], we design an algorithm to compute a regular isotopic ε -approximation of an n -ellipse. We also augment the algorithm to compute simultaneous contour plots of the distance function φ , resulting in a useful visualization tool (see Figure 1).

- We implement and experimentally evaluate the performance of all our algorithms on different datasets in the plane, as a function of n and ε . Various details of the interval primitives and proofs can be found in the full arXiv version.

2 Preliminaries

Vector variables are written in bold font: thus $\mathbf{0}$ is the origin of \mathbb{R}^d and $\mathbf{x} = (x_1, \dots, x_d)$. For a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, let $\partial_i f$ denote partial differentiation with respect to x_i . The *gradient* $\nabla f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ of f is given by the vector $\nabla f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_d(\mathbf{x}))^T$ where $f_i = \partial_i f$. In general, the operator ∇ is partial, i.e., $\nabla f(\mathbf{x}_0)$ might not be defined at a point \mathbf{x}_0 . A point \mathbf{x}_0 is a *critical point* of f if $\nabla f(\mathbf{x}) = \mathbf{0}$ or $\nabla f(\mathbf{x})$ is undefined.

We consider analytic properties of a scalar function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, mainly from the viewpoint of convex analysis [35, 39]. In our case, f is the Fermat distance function for some weighted set A . From an abstract perspective, the Fermat point problem (resp., n -ellipsoid problem) amounts to computing the critical points of the gradient of f (resp., computing the level sets of f). The Fermat point is the only critical point of ∇f in $\mathbb{R}^d \setminus A$, assuming A is non-collinear.

Most of the basic properties regarding the Fermat point are well-known and may be found in our references such as [32, 35, 39, 43, 58]. To emphasize the foci set A , we explicitly write φ_A instead of φ . A focus $\mathbf{a} \in A$ is the Fermat point of A if and only if $\|\nabla \varphi_{A \setminus \mathbf{a}}(\mathbf{a})\| \leq w(\mathbf{a})$. Testing if the Fermat point \mathbf{x}^* is in A can be done in $O(n^2 d)$ time. If \mathbf{x}^* is not one of the foci, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$, and the problem can be reduced to general finding real zeros of a square system of polynomial equations (e.g., [59]). However, the thrust of this paper is to develop direct methods that exploit the special properties of the Fermat problem.

We formally define the two main problems which we consider:

- APPROXIMATE FERMAT POINT: Given a weighted point set A in \mathbb{R}^d and $\varepsilon > 0$, compute a point $\tilde{\mathbf{x}}^*$ within ε distance to the Fermat point \mathbf{x}^* of A .
- APPROXIMATE ISOTOPIC n -ELLIPSES: Given $\varepsilon > 0$, a weighted point set A in \mathbb{R}^2 of size n and a radius $r > r^*(A)$, compute a closed polygonal curve E that is ε -isotopic to $\varphi^{-1}(r)$, i.e., there exists an ambient isotopy¹ $\gamma : \mathbb{R}^2 \times [0, 1] \rightarrow \mathbb{R}^2$ with $\gamma(E, 1) = \varphi^{-1}(r)$ and for any point $\mathbf{a} \in \varphi^{-1}(r)$, the parametric curve $\gamma(\mathbf{a}, \cdot)$ has at most length ε . This implies a bound of ε on the Hausdorff distance between E and $\varphi^{-1}(r)$.

Approximation notions. We compare the three different notions of ε -approximation for the Fermat point. We reduce the approximation problem of notion (C) to (B), and (B) to (A). An ε -approximation $\tilde{\mathbf{x}}^*$ of \mathbf{x}^* in the sense $\|\tilde{\mathbf{x}}^* - \mathbf{x}^*\| \leq \varepsilon$ is also a $(W\varepsilon)$ -approximation in the sense $\varphi(\tilde{\mathbf{x}}^*) \leq \varphi(\mathbf{x}^*) + W\varepsilon$, which follows directly from the triangle inequality

$$\varphi(\tilde{\mathbf{x}}^*) = \sum_{\mathbf{a} \in A} w(\mathbf{a}) \|\tilde{\mathbf{x}}^* - \mathbf{a}\| \leq \sum_{\mathbf{a} \in A} w(\mathbf{a}) (\|\tilde{\mathbf{x}}^* - \mathbf{x}^*\| + \|\mathbf{x}^* - \mathbf{a}\|) = W\varepsilon + \varphi(\mathbf{x}^*).$$

An ε -approximation $\tilde{\mathbf{x}}^*$ of \mathbf{x}^* in the sense $\varphi(\tilde{\mathbf{x}}^*) \leq \varphi(\mathbf{x}^*) + \varepsilon$ is also a $\frac{2\varepsilon}{\varphi(g)}$ -approximation in the sense $\varphi(\tilde{\mathbf{x}}^*) \leq (1 + \frac{2\varepsilon}{\varphi(g)})\varphi(\mathbf{x}^*)$. The center of gravity g is a 2-approximation of the Fermat radius r^* (see [16]), i.e. $\varphi(\mathbf{x}^*) \geq \frac{1}{2}\varphi(g)$. Hence

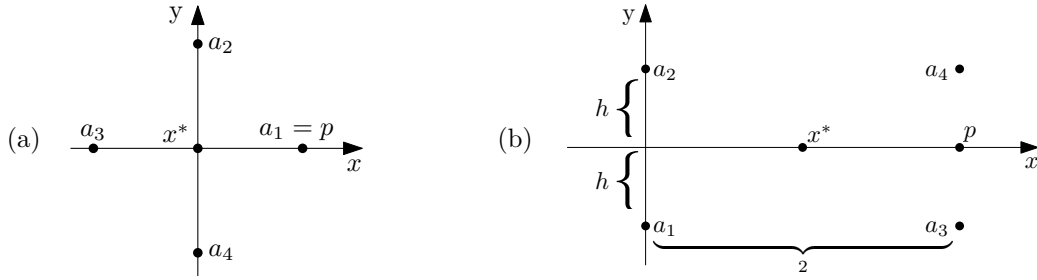
$$\varphi(\tilde{\mathbf{x}}^*) \leq \varphi(\mathbf{x}^*) + \varepsilon = \left(1 + \frac{\varepsilon}{\varphi(\mathbf{x}^*)}\right) \varphi(\mathbf{x}^*) \leq \left(1 + \frac{2\varepsilon}{\varphi(g)}\right) \varphi(\mathbf{x}^*)$$

¹ That is, a continuous map $\gamma : \mathbb{R}^2 \times [0, 1] \rightarrow \mathbb{R}^2$ such that $\gamma_0 = \gamma(\cdot, 0)$ is the identity map, and, for all $t \in [0, 1]$, $\gamma_t = \gamma(\cdot, t)$ is a homeomorphism on \mathbb{R}^2 .

On the other hand, it is not clear how to derive an ε -approximation of type (A) if an approximation algorithm for type (B) and (C) is at hand, as the following 2 examples show.

Example 1: For any $\varepsilon > 0$ choose $c \leq \frac{\varepsilon}{2\sqrt{2}-2}$ and consider the weighted foci $\mathbf{a}_1 = (1, 0)$, $\mathbf{a}_2 = (0, 1)$, $\mathbf{a}_3 = (-1, 0)$, $\mathbf{a}_4 = (0, -1)$ with $w(\mathbf{a}_1) = w(\mathbf{a}_3) = 1$ and $w(\mathbf{a}_2) = w(\mathbf{a}_4) = c$ for which the Fermat point is $\mathbf{x}^* = (0, 0)$ for symmetry reasons, see Figure 3(a). Point $p = (1, 0)$ is an ε -approximation of \mathbf{x}^* in the sense (B) and (C), but it has a distance of 1 to \mathbf{x}^* .

Example 2: For any $\varepsilon > 0$ we choose $h > 0$ small enough such that: $2\sqrt{4+h^2} + 2h \leq 4\sqrt{1+h^2} + \varepsilon$. Consider the foci $\mathbf{a}_1 = (0, -h)$, $\mathbf{a}_2 = (0, h)$, $\mathbf{a}_3 = (2, -h)$, $\mathbf{a}_4 = (2, h)$ with unit weights. The Fermat point is $\mathbf{x}^* = (1, 0)$ for symmetry reasons, see Figure 3(b). Point $p = (2, 0)$ is an ε -approximation of \mathbf{x}^* in the sense (B) and (C), but it has a distance of 1 to \mathbf{x}^* .



■ **Figure 3** (a) Example that a *good* approximation of the Fermat point in sense (B) does not imply a *good* approximation in sense (A). (b) Analogous example for sense (C).

Subdivision Paradigm. The subdivision algorithms presented in this paper take as input an initial box $B_0 \subset \mathbb{R}^d$ and recursively split it. We organize the boxes in a *generalized quadtree* data structure [50]. A box can be specified by d intervals as $B = I_1 \times I_2 \times \dots \times I_d$. Let m_B denote the *center* of B , r_B the *radius* of B (distance between m_B and a corner), and $\omega(B)$ the *width* of B (the maximum length of its defining intervals). The term $c \cdot B$ denotes the box with center m_B and radius $c \cdot r_B$. The function SPLIT_1 takes a box B and returns 2^d congruent subboxes (*children*), one for each orthant. We use SPLIT_2 to indicate that we do two successive levels of SPLIT_1 operations (i.e., $1 + 2^d$ SPLIT_1 operations, resulting in $(2^d)^2 = 4^d$ leaves).

Soft Predicates. Let $\square \mathbb{R}^d$ denote the set of closed d -dimensional boxes (i.e., Cartesian products of intervals) in \mathbb{R}^d . Let P be a logical *predicate* on boxes, i.e., $P : \square \mathbb{R}^d \rightarrow \{\text{true}, \text{false}\}$. For example, the *Fermat point predicate* is given by $P_{\text{fp}}(B) = \text{true}$ if and only if $\mathbf{x}^* \in B$. Logical predicates are hard to implement, and thus, we may focus on *tests*, which are viewed as one-sided predicates. Formally, a test T looks like a predicate: $T : \square \mathbb{R}^d \rightarrow \{\text{success}, \text{failure}\}$ and it is always associated to some predicate P : call T a *test for predicate P* if $T(B) = \text{success}$ implies $P(B) = \text{true}$. However, we conclude nothing if $T(B) = \text{failure}$. Denote this relation by “ $T \Rightarrow P$ ”.

Soft predicates [56] are an intermediate concept between a test and a predicate. Typically, they arise from a partial scalar function $f : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{\uparrow\}$ where $f(\mathbf{x}) = \uparrow$ means $f(\mathbf{x})$ is not defined. We then define a partial *geometric predicate* P_f on boxes B as follows:

$$P_f(B) = \begin{cases} \uparrow & \text{if } \uparrow \in f(B), \\ +1 & \text{if } f(B) > 0, \\ -1 & \text{if } f(B) < 0, \\ 0 & \text{else.} \end{cases}$$

We can now derive various logical predicates P from P_f , by identifying the values in the set $\{-1, 0, +1, \uparrow\}$ with **true** or **false**. For instance, we call P an *exclusion predicate* if we associate the 0- and \uparrow -value with **false** and the other values with **true**. For the *inclusion* predicate, we associate the 0-value with **true**, others with **false**. For example, a test for the Fermat point predicate P_{fp} is an inclusion predicate based on the partial function $f(\mathbf{x}) = \sum_i (\partial_i f(\mathbf{x}))^2$; the function is partial because $f(\mathbf{x}) = \uparrow$ when \mathbf{x} is a focus point. Although our box predicates $P(B)$ are defined for full-dimensional boxes B , we can extend them to any point \mathbf{x} as follows: $P(\mathbf{x})$ has the logical value associated with the $\text{sign}(f(\mathbf{x})) \in \{\uparrow, +1, -1, 0\}$.

► **Definition 1.** Let T be a test for a predicate P . We call T a *soft predicate* (or *soft version of P*) if it is convergent in this sense: if $(B_i : i = 0, 1, \dots)$ is a monotone sequence of boxes $B_{i+1} \subseteq B_i$ that converges to a point \mathbf{a} , then $P(\mathbf{a}) \equiv T(B_i)$ for i large enough.

Here, “ $P(\mathbf{a}) \equiv T(B_i)$ ” means $P(\mathbf{a}) = \text{true}$ if and only if $T(B_i) = \text{success}$. A soft version of $P(B)$ is usually denoted $\Box P(B)$. We note that soft versions of exclusion predicates are generally easier to construct than inclusion predicates. The former can be achieved by numerical approximation, while the latter requires some deeper principle such as the Brouwer fixed point theorem [9].

Interval arithmetic. We construct soft predicates using functions of the form $F : \Box\mathbb{R}^d \rightarrow \Box(\mathbb{R} \cup \{-\infty, \infty\})$ that approximate the scalar function $f : D \rightarrow \mathbb{R}$ with $D \subset \mathbb{R}^d$.

► **Definition 2.** Call F a *soft version of f* if it is

- i) conservative, i.e., for all $B \in \Box\mathbb{R}^d$, $F(B)$ contains $f(B) := \{f(p) : p \in B \cap D\}$, and
- ii) convergent, i.e., if for monotone sequence $(B_i : i \geq 0)$ that converges to a point $\mathbf{a} \in D$, $\lim_{i \rightarrow \infty} \omega(F(B_i)) = 0$ holds.

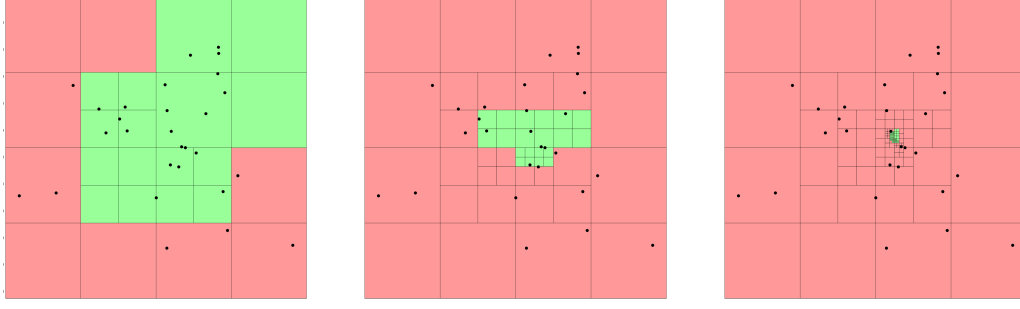
We shall denote F by $\Box f$ when F is a soft version of f . There are many ways to achieve $\Box f$. For example, if f has an arithmetic expression E , we can simply evaluate E using interval arithmetic. More sophisticated methods may be needed for performance. The next lemma shows how $\Box f$ leads to soft exclusion predicates based on f .

► **Lemma 3.** If P is an exclusion predicate based on f , then the test $\Box P(B) : 0 \notin \Box f(B)$ is a soft version of P .

Below, we need a multivariate generalization, to the case where $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^m$, and the exclusion predicate $P(B)$ is $\mathbf{0} \notin \mathbf{f}(B)$. If $\Box \mathbf{f} : \Box\mathbb{R}^d \rightarrow \Box\mathbb{R}^m$ is a soft version of \mathbf{f} , then a soft version of $P(B)$ is the given by the test $T(B) : \mathbf{0} \notin \Box \mathbf{f}(B)$. If $\mathbf{f} = (f_1, \dots, f_m)$, then this reduces to $0 \notin \Box f_i(B)$ for some $i = 1, \dots, m$.

3 Approximate Fermat points

We now present three approximation algorithms for the Fermat point \mathbf{x}^* . For simplicity, we assume in our algorithms that the Fermat point is not a focus, i.e. $\mathbf{x}^* \notin A$. This assumption can be easily checked in $O(n^2d)$ preprocessing time, or with a more elegant approach, in $O(nd)$ time during the execution of our subdivision algorithms.



■ **Figure 4** Different steps during the the execution of Algorithm 1. The dark red boxes cannot contain the Fermat point, whereas the light green boxes may contain it.

3.1 Using the Subdivision Paradigm

The subdivision paradigm requires an initial box B_0 to start subdividing. If B_0 is not given, it is easy to find a box that contains \mathbf{x}^* , since \mathbf{x}^* lies in the convex hull of A [32]. We use a function INITIAL-BOX(A) which, in $O(nd)$ time, computes an axis-aligned bounding box with corners having the minimum and maximum x, y coordinates.

We define an exclusion and inclusion predicate based on the gradient function $\nabla\varphi$.

► **Definition 4.** *Given a box B , the gradient exclusion predicate $C_0^\nabla(B)$ is defined by the condition $\mathbf{0} \notin \nabla\varphi(B)$. The gradient inclusion predicate $C_1^\nabla(B)$ is just the complement of $C_0^\nabla(B)$, that is $\mathbf{0} \in \nabla\varphi(B)$.*

Under our assumptions that $\mathbf{x}^* \notin A$, we have that $C_1^\nabla(B)$ holds if and only if $\mathbf{x}^* \in B$. We obtain a soft version of the exclusion predicate $C_0^\nabla(B)$ by replacing $\nabla\varphi$ in its definition with any soft version $\square\nabla\varphi$, see Lemma 3. But it is not so easy to get a soft version of $C_1^\nabla(B)$; we shall return to this when we treat the Newton operator below.

In Algorithm 1, using the exclusion predicate we discard boxes that are guaranteed not to contain \mathbf{x}^* (red in Figure 4) and we split boxes that might contain \mathbf{x}^* (green in Figure 4). While subdividing, we test whether we can already approximate \mathbf{x}^* well enough by putting a bounding box around all the boxes that are not excluded yet, using the following predicate.

► **Definition 5.** *Given a set of boxes Q that contains the Fermat point, the stopping predicate $C^\varepsilon(Q)$ returns true, if and only if the minimum axis-aligned bounding box containing all boxes in Q has a radius at most ε .*

If C^ε returns true, then we can stop. Since the radius of the minimum bounding box is at most ε , the center of the box is an ε -approximate Fermat point $\tilde{\mathbf{x}}^*$.

■ **Algorithm 1** Subdivision for the approximate Fermat point (SUB).

Input : Foci set A , constant $\varepsilon > 0$

Output: Point $\tilde{\mathbf{x}}^*$

```

1  $B_0 \leftarrow \text{INITIAL-BOX}(A); \quad Q \leftarrow \text{QUEUE}(); \quad Q.\text{PUSH}(B_0);$ 
2 while not  $C^\varepsilon(Q)$  do
3    $B \leftarrow Q.\text{POP}();$ 
4   if not  $\square C_0^\nabla(B)$  then
5      $Q.\text{PUSH}(\text{SPLIT}_1(B));$ 
6 return  $\tilde{\mathbf{x}}^* \leftarrow \text{Center of the bounding box of } Q;$ 

```

Regarding the runtime of Algorithm 1, evaluating $\nabla\varphi$ and its soft version takes linear time in n . The subdivision approach induces an exponential dependency on d , as splitting a box creates 2^d many children. Further, a SPLIT_1 operation decreases the boxwidth by a factor of 2, therefore, Algorithm 1 cannot converge faster than linear in ε .

3.2 Enhancing the Subdivision Paradigm

In this section, we augment Algorithm 1 with a speed up based on a *Newton operator*, which will ensure eventual quadratic convergence.

The Newton operator. Newton-type algorithms have been considered in the past, usually independently of other methods, and thus suffer from lack of global convergence. Moreover, from a numerical viewpoint, such methods face the *precision-control problem*. Our algorithm integrates subdivision with the Newton operator (an old idea that goes back to Dekker [17] in the 1960's), thus ensuring global convergence.

We want to find the Fermat point, i.e., the root of $\mathbf{f} = \nabla\varphi$. Newton-type operators are well-studied in the interval literature, and they have the form $N = N_{\mathbf{f}} : \square\mathbb{R}^d \rightarrow \square\mathbb{R}^d$. There are three well-known versions of $N_{\mathbf{f}}$: the simplest version, from Moore [36] and Nickel [40], is

$$N(B) = m_B - J_{\mathbf{f}}^{-1}(B) \cdot \mathbf{f}(m_B),$$

where $J_{\mathbf{f}}$ is the Jacobian matrix of \mathbf{f} . Since $\mathbf{f} = \nabla\varphi$, this matrix is actually the Hessian of φ . The second version by Krawczyk [31, 52] is:

$$N(B) = m_B - K \cdot \mathbf{f}(m_B) + (I - K \cdot \mathbf{f}(B)) \cdot (B - m_B),$$

where K is any non-singular $d \times d$ matrix, usually chosen to be an approximation of $J_{\mathbf{f}}^{-1}(m_B)$. The third version, from Hansen and Sengupta [24, 25], can be viewed as a sophisticated implementation of the Moore-Nickel operator using an iteration reminiscent of the Gauss-Seidel algorithm, combined with preconditioning. Later we report on our implementation of the first two Newton operators. In general, the Newton operator $N(B)$ does not return a box even if B is a box; so we define $\square N(B)$ to be a box that contains $N(B)$. For simplicity, we assume that $\square N(B)$ is the smallest box containing $N(B)$ with the same aspect ratio as B .

The following three properties of Newton box operators are consequences of Brouwer's Fixed Point Theorem [9, 41, 52, 59]:

1. (Inclusion Property) If $N(B) \subseteq B$ then $\mathbf{x}^* \in N(B)$.
2. (Exclusion Property) If $N(B) \cap B = \emptyset$ then $\mathbf{x}^* \notin B$.
3. (Narrowing Operator) If $\mathbf{x}^* \in B$ then $\mathbf{x}^* \in N(B)$.

Based on these properties, we can define two tests and an operator:

► **Definition 6.** *Newton tests for gradient exclusion/inclusion predicates (below we explain why we use $2B$ instead of B):*

- **Newton exclusion test:**
 $T_0^N(B) = \text{success}$ iff $N(2B) \cap B = \emptyset$. Thus $T_0^N(B) \Rightarrow C_0^{\nabla}(B)$.
- **Newton inclusion test:**
 $T_1^N(B) = \text{success}$ iff $N(2B) \subseteq 2B$. Thus $T_1^N(B) \Rightarrow C_1^{\nabla}(2B)$.
- **Newton narrowing operator:**
 $N_{\cap}(B)$ returns $B \cap N(2B)$.

Note that the Newton tests $T_0^N(B)$ and $T_1^N(B)$ are defined using the exact Newton operator $N(B)$. If we replace it by a soft version $\Box N(B)$ in these definitions, they remain as inclusion/exclusion tests for $C_1^\nabla(B)$ and $C_0^\nabla(B)$; we denote them by $\Box C_1^\nabla(B)$ and $\Box C_0^\nabla(B)$.

To compute $\Box N(B)$, we use standard interval arithmetic to evaluate the Newton operators. We already noted that if $N(B) \subseteq B$, then $\mathbf{x}^* \in N(B)$. But if \mathbf{x}^* is on the boundary of B , then $\Box N(B) \subseteq B$ might not hold, and this issue persists even after splitting B . We circumvent this problem by using $2B$ instead of B in the definition of $T_1^N(B)$.

We enhance Algorithm 1 by the soft inclusion predicate $\Box T_1^N(B)$, as sketched in Algorithm 2. If $\Box T_1^N(B)$ succeeds, we conclude that \mathbf{x}^* is contained in $\Box N(2B)$. In that case, we can discard all other boxes and initialize a new queue Q on $\Box N(2B)$. In subsequent calls to $\Box T_1^N(B')$ for $B' \in Q$, we conclude that $\mathbf{x}^* \in 2B'$. But to ensure that $w(2B') < w(B)$ (to avoid an infinite loop), we initialize the queue Q with the 4^d boxes of $\text{SPLIT}_2(\Box N(2B))$.

■ **Algorithm 2** Enhanced subdivision for the approximate Fermat point (*ESUB*).

As in Algorithm 1 but replace line 5 with the following:

5.1	if $\Box T_1^N(B)$ then	
5.2	$Q \leftarrow \text{QUEUE}();$	// initialize a new queue
5.3	$Q.\text{PUSH}(\text{SPLIT}_2(\Box N(2B)));$	// 2 SPLIT operations
5.4	else	
5.5	$Q.\text{PUSH}(\text{SPLIT}_1(B));$	

With respect to the runtime of Algorithm 2, we observe that once the soft Newton inclusion predicate succeeds, then it will also do so for an initial box of the new queue. This, essentially, divides the algorithm into two phases. The first phase can be basically seen as Algorithm 1. In the second phase, the Newton test guarantees quadratic convergence in ε . Getting into the second phase depends on the configuration of the foci set but not on ε , hence, our approach is of particular interest for small values of ε .

The termination of both subdivision algorithms follows from the soft gradient exclusion predicate being convergent. The algorithms terminate once the predicate $C^\varepsilon(Q)$ succeeds, yielding an ε -approximate Fermat point, so we summarize as follows.

► **Theorem 7.** *Both Algorithms 1 and 2 terminate and return an ε -approximate Fermat point.*

3.3 Certifying the Weiszfeld method

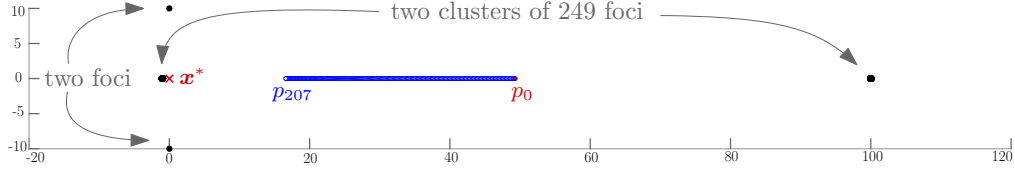
Weiszfeld's iterative method [32, 43, 58] describes a sequence \mathbf{p}_i ($i = 0, 1, \dots$) of points that converges to the Fermat point \mathbf{x}^* , starting from any initial \mathbf{p}_0 . The recurrence relation is $\mathbf{p}_{i+1} = T(\mathbf{p}_i)$, where $T(\mathbf{x})$ is defined by

$$T(\mathbf{x}) = \frac{\sum_{\mathbf{a} \in A, \mathbf{a} \neq \mathbf{x}} w(\mathbf{a}) \frac{\mathbf{a}}{\|\mathbf{x} - \mathbf{a}\|}}{\sum_{\mathbf{a} \in A, \mathbf{a} \neq \mathbf{x}} w(\mathbf{a}) \frac{1}{\|\mathbf{x} - \mathbf{a}\|}}.$$

Note that when \mathbf{x} is a focus, then $T(\mathbf{x})$ depends just on all other foci.

This simple iterative method is widely used, and although it converges, it does not solve our ε -approximation problem as we do not know when to stop. To see that this is a real issue, consider the example in Figure 5.

We augment the Weiszfeld iteration by adding Newton tests during the computation, turning it into an ε -approximation algorithm. While at the i -th iteration, we define a small box B with point \mathbf{p}_i as center, and map it to the box $\Box N(B)$ using the Newton operator; see Figure 6. If $\Box N(B) \subseteq B$, then the Fermat point \mathbf{x}^* lies in $\Box N(B)$. On the contrary, if $\Box N(B) \not\subseteq B$ we move on to the next point \mathbf{p}_{i+1} and adjust the box size as follows.



■ **Figure 5** An example with 500 foci, showing that Weiszfeld's scheme does not solve the ε -approximation problem. The scheme stopped when $\|p_{i-1} - p_i\| \leq 1/10$, after 207 steps (blue points). The distance $\|x^* - p_{207}\|$ can be arbitrarily big ($\|x^* - p_{207}\| > 15$ in this case).

If $\frac{B}{10} \cap \square N(\frac{B}{10}) = \emptyset$, then the box $\frac{B}{10}$ does not contain x^* and we therefore expand B by a factor of 10. If $\frac{B}{10} \cap \square N(\frac{B}{10}) \neq \emptyset$, then there might be a focus in box $\frac{B}{10}$, which hinders $\square N(B) \subseteq B$ to succeed. In that case we shrink B by a factor of 10. If a focus is not in $\frac{B}{10}$, shrinking B does not effect the algorithm negatively, as B can expand again.

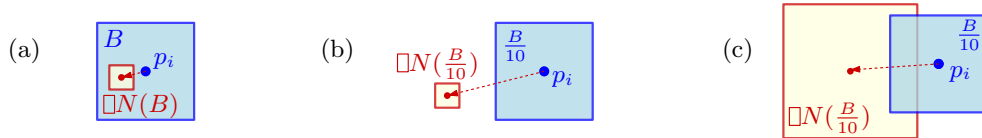
Using these tests we augment the point sequence scheme, sketched in Algorithm 3, with the property that if the Newton test evaluates to true, then we are guaranteed an ε -approximation of x^* . As a starting point, we choose the *center of mass* p_0 of A , i.e., $p_0 = \frac{1}{W} \sum_{a \in A} w(a) a$.

With respect to the runtime, the point sequence $T(x)$ converges linearly in ε towards x^* [29] but in order for Algorithm 3 to terminate the test $\square N(B) \subseteq B$ must succeed. Similar to other Newton operators, $\square N(B) \subseteq B$ succeeds for boxes in a neighborhood surrounding x^* . This neighborhood depends only on the configuration of A but not on ε . Further, evaluating $T(x)$ and $\square N(B)$ can be done in $O(nd^2)$ time. We conclude as follows.

► **Theorem 8.** *Algorithm 3 terminates and returns an ε -approximate Fermat point.*

■ **Algorithm 3** Certified Weiszfeld for the approximate Fermat point (CW).

Input: Foci set A , constant $\varepsilon > 0$	Output: Point \tilde{x}^*
1 $p \leftarrow p_0$; $l \leftarrow \varepsilon$;	
2 while $TRUE$ do	
3 $B \leftarrow \text{Box } B(m_B = p, \omega(B) = l)$;	
4 if $\square N(B) \subseteq B$ then	// Figure 6(a)
5 return $\tilde{x}^* \leftarrow p$;	
6 else if $\square N(\frac{B}{10}) \cap \frac{B}{10} = \emptyset$ then	// Figure 6(b)
7 $l \leftarrow \min\{10 \cdot l, \varepsilon\}$;	
8 else	// Figure 6(c)
9 $l \leftarrow \frac{1}{10} \cdot l$;	
10 $p \leftarrow T(p)$;	



■ **Figure 6** The case analysis of Algorithm 3. (a) $\square N(B) \subseteq B$, (b) $\square N(\frac{B}{10}) \cap \frac{B}{10} = \emptyset$, and (c) $\square N(\frac{B}{10}) \cap \frac{B}{10} \neq \emptyset$.

4 Approximating n -ellipses

In this section, we describe an algorithm to construct approximate n -ellipses, based on the subdivision paradigm. Throughout this work we maintain the subdivision *smooth*, i.e., the width of any two adjacent boxes, which are leaves of the quadtree, may differ at most by a factor of 2. Maintaining smoothness is easy to implement and has amortized $O(1)$ cost per operation [6]. Without maintaining smoothness, the amortized cost can be $\Omega(\log n)$ [6].

The Plantinga and Vegter (PV) construction [47, 33, 34] approximates the zero set of a function $F : \mathbb{R}^d \rightarrow \mathbb{R}$ where $d \in \{2, 3\}$. Assuming that $S = F^{-1}(0)$ is regular, i.e., the gradient ∇F is non-zero at every point of S , this approximation is isotopic to S . Our goal is to use this construction to approximate the n -ellipse defined by $F(p) := \varphi(p) - r$ with $r > r^*$. For simplicity, we assume all boxes are square; for the construction to succeed, we only need an aspect ratio $\leq \sqrt{2}$ (see [33]). We use the notation $\langle \cdot, \cdot \rangle$ for the scalar product. The following are the key predicates and tests in the PV construction of the n -ellipse $F^{-1}(0)$.

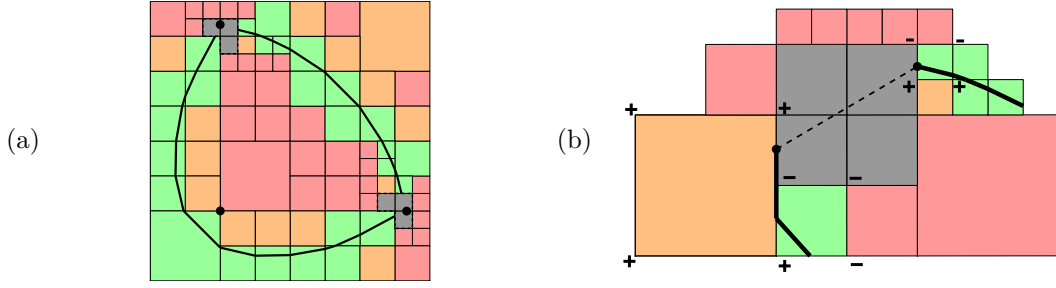
► **Definition 9.** Fix $F(p) = \varphi(p) - r$. Let B be a square box.

1. The fundamental box predicate is the inclusion predicate $C_1^F(B) : 0 \in F(B)$, and its complement, the exclusion predicate $C_0^F(B) : 0 \notin F(B)$.
2. The (corner) inclusion test $T_{\text{cor}}(B) = \text{success}$ iff F , when evaluated at the corners of B , admits both negative and positive values. Clearly, $T_{\text{cor}}(B)$ is a test for $C_1^F(B)$. (There is a standard PV trick whereby any 0-value can be arbitrarily made positive.)
3. The normal variation predicate $C_{\text{nv}}(B)$ is defined by the condition $\langle \nabla F(B), \nabla F(B) \rangle > 0$.

We obtain the soft versions $\Box C_0^F(B)$ and $\Box C_{\text{nv}}(B)$ by the usual device of replacing $F(B)$ in the definition of the predicates by a soft version $\Box F(B)$. But for the inclusion predicate $C_1^F(B)$ we have no soft version. Instead, the corner test $T_{\text{cor}}(B)$ is a test for $C_1^F(B)$. To supplement the corner test, we need the normal variation predicate $C_{\text{nv}}(B)$. This predicate is equivalent to the condition that the angle between the gradient of any two points in B is at most 90° . It implies that the n -ellipse is monotone in either x - or y -direction within the box. In Figure 7, boxes are: **red** if they pass the $\Box C_0^F(B)$ test, **green** if they pass both $\Box C_{\text{nv}}$ and T_{cor} , **orange** if they pass only $\Box C_{\text{nv}}$, and **gray** otherwise. Note that orange boxes may, or may not, contain parts of the approximate n -ellipse.

An n -ellipse is not regular if it passes through some focus [51]; in that case a direct PV construction is not possible. We develop a variation, sketched in Algorithm 4, where we simultaneously subdivide boxes and construct pieces of the n -ellipse *on the fly*, instead of doing that in the end. Further, boxes in which the n -ellipse may not be regular are treated differently. During the subdivision part of the algorithm, we classify boxes in three categories:

1. Boxes which satisfy $\Box C_0^F(B)$ (**red**): These do not contain any piece of the n -ellipse, so they do not need to be further considered and are discarded.
2. Boxes which satisfy $\Box C_{\text{nv}}$ and have width smaller than $\varepsilon/2$ (**green** or **orange**): We immediately draw edges in each of these boxes, in contrast to the normal PV construction. Note that at a later stage of the algorithm it might happen that we split one of B 's neighboring boxes. In that case we need to take into account the sign of F at the new vertex on B 's boundary. If necessary, the edges in box B then need to be updated.
3. The remaining boxes (**gray**): Such boxes occur near foci and need more careful attention, as we cannot apply the standard PV construction. Instead, given a set of gray boxes we first distinguish them in connected components, using a DFS algorithm. Then, for each connected component of gray boxes K_i , we check if a set of conditions is satisfied:



■ **Figure 7** (a) A 3-ellipse passing through two foci. Components of gray boxes (temporarily) surround the foci. (b) If a gray component satisfies (B1) - (B3) the two ingoing edges are connected with an edge (shown dashed).

(B1) K_i contains exactly one focus.

(B2) There are exactly two PV-edges leading to K_i .

(B3) The distance between any two corners of the boxes in K_i is at most $\varepsilon/2$.

If K_i satisfies all (B1) - (B3), then we connect the 2 PV-edges leading to K_i by a line segment and discard boxes of K_i , see Figure 7(b). Otherwise, the children of the boxes of K_i are put back in Q for further classification.

■ **Algorithm 4** Approximating an n -ellipse.

Input: Foci set A , radius r , constant ε , box B_0	Output: Curve E
--	--------------------------

```

1  $Q \leftarrow \text{QUEUE}(); \quad Q.\text{PUSH}(B_0);$ 
2 while  $Q \neq \emptyset$  do
3    $Q_{\text{new}} \leftarrow \text{QUEUE}();$ 
4   while  $Q \neq \emptyset$  do
5      $B \leftarrow Q.\text{POP}();$ 
6     if not  $\square C_0^F(B)$  then                                     // exclude red
7       if  $\square C_{\text{nw}}(B)$  and  $\omega(B) < \varepsilon/2$  then                 // green or orange
8          $E \cap B \leftarrow \text{ONLINE-PV}(B);$ 
9       else                                                         // gray
10         $Q_{\text{new}}.\text{PUSH}(\text{SPLIT}_4(B));$ 
11    $Q \leftarrow \text{CONNECTED-COMPONENTS-ANALYSIS}(Q_{\text{new}})$ 
12 return  $E;$ 
```

By controlling the size of the boxes containing parts of the output curve, and by the modification the PV construction we prove the following.

► **Theorem 10.** *Algorithm 4 returns an isotopic ε -approximation of the n -ellipse $F^{-1}(0)$.*

Interpolating edges. The PV construction creates edges within a box B , which start and end at midpoints of box edges. One can derive a nicer-looking approximation by using linear interpolation on the box edges by taking into account the value of F at B 's corners.

Contour Plotting. As an application, we can use the above technique in order to produce a topologically correct, ε -approximate and visually nice n -elliptic contour plot. To do so, we first adapt our algorithm in order to simultaneously plot several n -ellipses inside a bounding

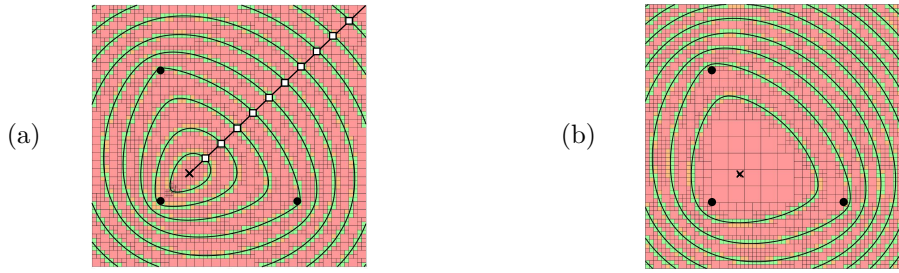


Figure 8 Two different 3-elliptic contour plots with 10 contour lines, having the same set of foci. (a) Using radii of *equidistant points*. (b) Using *equidistant radii*.

box, corresponding to the same foci but with different radii. Each n -ellipse is a *contour line*, and we describe how to plot them *visually nice*, i.e., the contour lines are roughly equally distributed in space. See Figure 8 for two different approaches and their visualization effect.

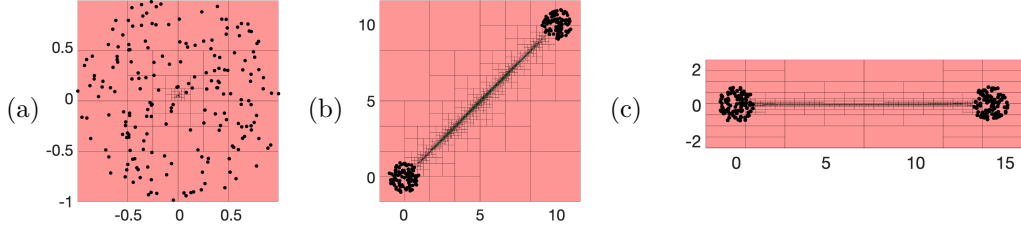
5 Experiments

We implemented our algorithms for \mathbb{R}^2 and conducted a series of experiments. Our current software is written in MATLAB (version R2018b), taking advantage of its graphics ability. The numerical accuracy is therefore IEEE numerical precision. The platform used was MacOS Big Sur v11.2.3, with 2.5 GHz Quad-Core Intel Core i7 and 16 GB 1600MHz DDR3.

Following, we report on our experiments, discussing some notable points one by one. We evaluated our algorithms on both synthetic and real-world datasets. For all algorithms approximating the Fermat point we chose a time limit of 600 seconds. Moreover, for most experiments we executed 10 different instances for completeness. In the illustrated charts, the curves pass through the mean of the 10 running times, and additionally we also marked the minimum and maximum running times. All axes in the charts are of logarithmic scale.

Datasets. We mainly experimented with two different types of synthetic datasets, namely UNIF-1 and UNIF-2. In UNIF-1 the n foci are sampled uniformly from a disk of radius 1. In UNIF-2 again the n foci are sampled uniformly from a disk of radius 1 and then $n/2$ foci are translated by a vector $(10, 10)$, see Figure 9(a) and Figure 9(b). Despite their similarity, the two datasets present strong differences. As we later see, UNIF-2 is significantly more difficult to solve in comparison to UNIF-1, and further UNIF-1 resembles nicely real-world datasets. The foci of UNIF-2 lie almost all on a common line, which implies that there are many points for which the gradient is close to 0. This makes it difficult to find the actual Fermat point, for which the gradient is exactly 0. We experimented with more types of synthetic datasets, such as points in convex position, vertices of a regular n -gon, clusters of points, but we do not report on these results, as they are similar to UNIF-1 or UNIF-2.

Newton operators. Adding a Newton operator to the subdivision process drastically improves the running time. We compared Algorithm 1 with two versions of Algorithm 2, where we once use the Newton operator based on Moore and Nickel and also the operator by Krawczyk. The results for various values of n and ε on both UNIF-1 and UNIF-2 are summarized in Figure 10. Note that Algorithm 2 initially needs to perform simple splitting operations until at some point the Newton test succeeds the first time. After that the algorithm converges quadratically in ε , which explains why the running time of both versions almost do not increase for decreasing ε . Even though the operator by Krawczyk returns a



■ **Figure 9** A box subdivision for $n = 200$ foci: (a) UNIF-1, (b) UNIF-2 and (c) UNIF-2 after PCA.

smaller box $\square N(B)$, i.e. it is more precise, than Moore and Nickel, it performs slower for UNIF-1 as evaluating the operator takes more time. We conclude that using a Newton operator speeds up the computations, and we use the one of by Moore and Nickel in Algorithm 2.

Principal component analysis. Foci sets like UNIF-2 are challenging as all foci are close to a common line. In this case, the subdivision algorithms can be slow because there are many boxes for which the gradient $\nabla\varphi$ is close to 0. Our approach to tackle this problem is to use subdivision with rectangular boxes. In a preprocessing step we do a *principal component analysis* (PCA) of the foci as heuristic. Then, we rotate the coordinate system such that the x -direction is the first principal component. In the box subdivision we use rectangular boxes with long x -width, see Figure 9(c). Observe in the following table, that for well distributed foci sets like UNIF-1, using the PCA adds only a small overhead to the total running time.

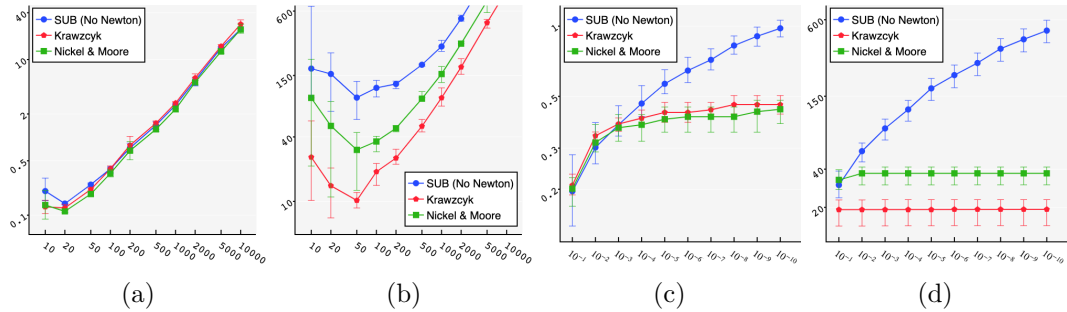
$\varepsilon = 10^{-3}, n =$	10	100	1000	10000
without PCA	0.12	0.31	2.33	23.4
with PCA	0.10	0.30	2.30	23.9

$n = 100, \varepsilon =$	10^{-1}	10^{-3}	10^{-5}	10^{-7}
without PCA	0.20	0.30	0.33	0.34
with PCA	0.18	0.30	0.33	0.35

On the contrary, for sets like UNIF-2, adding the PCA decreases drastically the running time, as shown next. Hence, the PCA preprocessing is a useful addition to Algorithm 2, which we will use also in the following experiments.

$\varepsilon = 10^{-3}, n =$	10	100	1000	10000
without PCA	90.7	48.5	170	timeout
with PCA	0.15	0.40	3.21	32.7

$n = 100, \varepsilon =$	10^{-1}	10^{-3}	10^{-5}	10^{-7}
without PCA	37.1	49.2	49.2	49.5
with PCA	0.36	0.40	0.42	0.43



■ **Figure 10** A comparison of Algorithm 1 (• SUB), Algorithm 2 with the Krawczyk Newton operator (• Krawczyk), and Algorithm 2 with the Nickel and Moore Newton operator (■ Nickel & Moore). (a),(b) Time as a function of n , with $\varepsilon = 10^{-4}$. (c),(d) Time as a function of ε with $n = 100$. (a),(c) UNIF-1 datasets. (b),(d) UNIF-2 datasets.

Real Datasets. Inspired by the applications in facility location we chose to experiment with instances of the well-known *Traveling Salesman Problem Library* [49] or TSPLIB. The foci correspond mostly to location of cities in different areas around the world. It appears that real-world instances show a similar behavior to UNIF-1 datasets; so we infer that UNIF-1 are realistic datasets for the evaluation of different algorithms. In our experiments, to verify that for each TSPLIB dataset we created an additional foci set, where we uniformly sampled the same number of foci in the axis-aligned bounding box. As ε we chose 10^{-6} times the width of the corresponding bounding box. These experiments are illustrated in Figure 12(a), and the similarity of the running time for the two datasets is obvious.

Summary on the Fermat point. We make an overall comparison of Algorithm 1, Algorithm 2 with the PCA, and Algorithm 3, illustrated in Figure 11. The running time of all methods shows a linear dependency on n , but there are big differences regarding the dependency on ε . Overall, Algorithm 3 performs well in all cases, but due to the linear convergence of Weiszfeld's point sequence, it cannot converge faster as ε decreases. In contrast, Algorithm 2 takes more time in the subdivision phase, but once the Newton tests succeeds, the algorithm terminates very quickly. So, it does not exhibit almost any changes in the running time for decreasing ε . This makes it favorable when a high precision approximate solution is required. It is also very fast in UNIF-2 instances and outperforms Algorithm 3. Summarizing, we suggest to use Algorithm 2 in small dimensional spaces and for small ε due to its eventual quadratic convergence in ε . On the other hand, the subdivision methods take exponential time in d , therefore, we suggest to use Algorithm 3 for higher dimensional spaces.

n -ellipses. Finally, we evaluated the runtime of n -ellipses algorithm. In Figure 12(b) we evaluate the dependency on n . In order to keep the length of the curve almost constant we choose the radii $r = \frac{(10\sqrt{2}+2)n}{2}$. The bounding box used is $[-2, 12]^2$. In Figure 12(c) we analyze the dependency on the length of the n -ellipse. The bounding box is fixed and we experimented with different radii such that the lengths of the curve differ by a factor of $3/2$. The runtime shows a linear dependency on n , as expected, and it also shows a linear dependency on the length of curve. This can be justified, as covering an n -ellipse of length l with boxes of width ε takes $O(l/\varepsilon)$ many boxes.

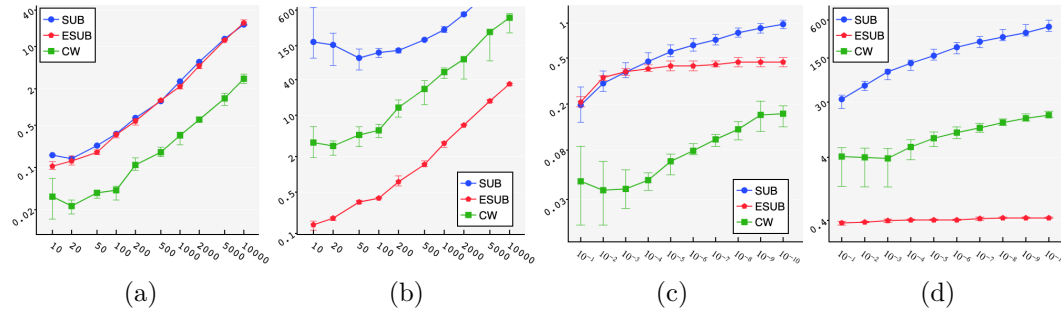


Figure 11 An overall comparison of Algorithm 1 (● SUB), Algorithm 2 with the PCA (◆ ESUB), and Algorithm 3 (■ CW). (a),(b) Time as a function of n , with $\varepsilon = 10^{-4}$. (c),(d) Time as a function of ε with $n = 100$. (a),(c) UNIF-1 datasets. (b),(d) UNIF-2 datasets.

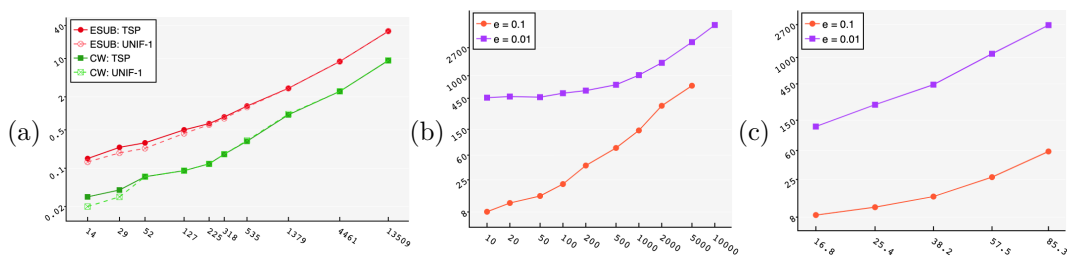


Figure 12 (a) A comparison of TSP data sets (filled shapes) with UNIF-1 (empty shapes, dashed curve) for both Algorithm 2 (● ESUB) and Algorithm 3 (■ CW). Fermat point with time as a function of n . (b),(c) n -ellipse on UNIF-2 with time as a function of (b) n and (c) the length of the n -ellipse. Two ε approximations with $\varepsilon = 0.1$ (●) and $\varepsilon = 0.01$ (■) have been computed.

6 Concluding Remarks

In this work, we mainly focused on finding ε -approximate Fermat points, in a strong sense $\|\tilde{\mathbf{x}}^* - \mathbf{x}^*\| \leq \varepsilon$, which had not been considered before. This approximation can also be used to derive an ε -approximation of the Fermat radius. This was done using a simple-to-implement subdivision approach. All of our algorithms are certified in the sense of interval arithmetic. Moreover, we certified the famous point-sequence algorithm of Weiszfeld [58] to guarantee that it does find an ε -approximate Fermat point. We also designed an algorithm to construct ε -approximate n -ellipses. The simplicity and efficiency of our algorithms were evaluated experimentally for $d = 2$.

There are many directions for further research. One is to derive algorithmic complexity bounds. Our intuition regarding the time complexity of our algorithms was affirmed by the experimental runtime evaluation. Such bounds are rare for iterative numerical algorithms. There has been considerable success in the area of root isolation [10, 11] where the idea of “continuous amortization” should also apply here. Further, we expect the usage of the Hansen-Sengupta Newton operator to result in a speedup.

Regarding the construction of n -ellipses, it would be interesting to design an alternative algorithm based on curve-tracing. This could improve the runtime once a starting point on the n -ellipse is found.

Another direction is related to *Voronoi diagrams*. From one perspective, it is interesting to approximate the Voronoi diagram, where the sites are n -ellipses; so far only 2-ellipses have been studied [19]. From a different perspective, if the sites are sets of foci (each associated with a Fermat distance function) it is interesting to compute their Voronoi diagram, defined as the minimization diagram of the Fermat distance functions. This is a *min-sum* diagram in the context of cluster Voronoi diagrams, see e.g., [28, 44]. We believe that subdivision methods augmented with root boxes, similar to [5], would be applicable to these problems.

References

- 1 A. Karim Abu-Affash and Matthew J. Katz. Improved bounds on the average distance to the Fermat-Weber center of a convex object. *Information Processing Letters*, 109(6):329–333, 2009.
- 2 Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proc. Symposium on Theory of Computing*, pages 250–257. ACM, 2002. doi:10.1145/509907.509947.
- 3 Chandrjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(2):177–191, 1988.

- 4 Amir Beck and Shoham Sabach. Weiszfeld's method: Old and new results. *Journal of Optimization Theory and Applications*, 164(1):1–40, 2015.
- 5 Huck Bennett, Evanthia Papadopoulou, and Chee Yap. Planar minimization diagrams via subdivision with applications to anisotropic Voronoi diagrams. *Computer Graphics Forum*, 35(5):229–247, 2016. doi:10.1111/cgf.12979.
- 6 Huck Bennett and Chee Yap. Amortized analysis of smooth quadrees in all dimensions. *Computational Geometry*, 63:20–39, 2017. doi:10.1016/j.comgeo.2017.02.001.
- 7 Bhaswar B. Bhattacharya. On the Fermat-Weber point of a polygonal chain and its generalizations. *Fundamenta Informaticae*, 107(4):331–343, 2011.
- 8 Prosenjit Bose, Anil Maheshwari, and Pat Morin. Fast approximations for sums of distances, clustering and the Fermat-Weber problem. *Computational Geometry*, 24(3):135–146, 2003.
- 9 Luitzen Egbertus Jan Brouwer. Über Abbildung von Mannigfaltigkeiten. *Mathematische Annalen*, 71(1):97–115, 1911.
- 10 Michael Burr, Felix Krahmer, and Chee Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. *Electronic Colloquium on Computational Complexity*, TR09(136), 2009.
- 11 Michael A. Burr. Continuous amortization and extensions: With applications to bisection-based root isolation. *Journal of Symbolic Computation*, 77:78–126, 2016. doi:10.1016/j.jsc.2016.01.007.
- 12 Paz Carmi, Sarel Har-Peled, and Matthew J. Katz. On the Fermat-Weber center of a convex object. *Computational Geometry*, 32(3):188–195, 2005. doi:10.1016/j.comgeo.2005.01.002.
- 13 Hui Han Chin, Aleksander Madry, Gary L. Miller, and Richard Peng. Runtime guarantees for regression problems. In *Proc. Innovations in Theoretical Computer Science*, pages 269–282. ACM, 2013.
- 14 Dietmar Cieslik. *Steiner minimal trees*, volume 23. Springer Science & Business Media, 2013.
- 15 Ernest J. Cockayne and Zdzislaw A. Melzak. Euclidean constructibility in graph-minimization problems. *Mathematics Magazine*, 42(4):206–208, 1969.
- 16 Michael B. Cohen, Yin Tat Lee, Gary L. Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proc. Symposium on Theory of Computing*, pages 9–21. ACM, 2016. doi:10.1145/2897518.2897647.
- 17 Theodorus Jozef Dekker. Finding a zero by means of successive linear interpolation. In *Constructive Aspects of the Fundamental Theorem of Algebra*, pages 37–48. Wiley Interscience, 1967.
- 18 Adrian Dumitrescu, Minghui Jiang, and Csaba D. Tóth. New bounds on the average distance from the Fermat-Weber center of a planar convex body. *Discrete Optimization*, 8(3):417–427, 2011. doi:10.1016/j.disopt.2011.02.004.
- 19 Ioannis Z. Emiris, Elias P. Tsigaridas, and George M. Tzoumas. The predicates for the Voronoi diagram of ellipses. In *Proc. Symposium on Computational Geometry*, pages 227–236. ACM, 2006.
- 20 Giovanni Francesco Fagnano. Problemata quaedam ad methodum maximorum et minimorum spectantia. *Nova Acta Eruditorum*, pages 281–303, 1775.
- 21 Sándor P. Fekete, Joseph S.B. Mitchell, and Karin Beurer. On the continuous Fermat-Weber problem. *Operations Research*, 53(1):61–76, 2005.
- 22 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proc. Symposium on Theory of Computing*, pages 569–578. ACM, 2011.
- 23 Horst Hamacher and Zvi Drezner. Facility location: applications and theory. *Science & Business Media: Springer*, 2002.
- 24 Eldon R. Hansen. A multidimensional interval newton method. *Reliable Computing*, 12(4):253–272, 2006. doi:10.1007/s11155-006-9000-y.
- 25 Eldon R. Hansen and Saumyendra Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.

- 26 Sarel Har-Peled and Akash Kushal. Smaller coresets for k -median and k -means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.
- 27 Sarel Har-Peled and Soham Mazumdar. On coresets for k -means and k -median clustering. In *Proc. 36th Annual ACM Symposium on Theory of computing*, pages 291–300. ACM, 2004.
- 28 Daniel P. Huttenlocher, Klara Kedem, and Micha Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete & Computational Geometry*, 9(3):267–291, 1993.
- 29 I. Norman Katz. Local convergence in Fermat’s problem. *Mathematical Programming*, 6(1):89–104, 1974.
- 30 Jakob Krarup and Steven Vajda. On Torricelli’s geometrical solution to a problem of Fermat. *Journal of Management Mathematics*, 8(3):215–224, 1997.
- 31 Rudolf Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4(3):187–201, 1969. doi:10.1007/BF02234767.
- 32 Harold W. Kuhn. A note on Fermat’s problem. *Mathematical programming*, 4(1):98–107, 1973.
- 33 Long Lin and Chee Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. *Discrete & Computational Geometry*, 45(4):760–795, 2011. doi:10.1007/s00454-011-9345-9.
- 34 Long Lin, Chee Yap, and Jihun Yu. Non-local isotopic approximation of nonsingular surfaces. *Computer-Aided Design*, 45(2):451–462, 2012.
- 35 Luis Fernando Mello and Lucas Ruiz dos Santos. On the location of the minimum point in the Euclidean distance sum problem. *São Paulo Journal of Mathematical Sciences*, 12:108–120, 2018.
- 36 Ramon E. Moore. *Interval Analysis*, volume 4. Prentice-Hall Englewood Cliffs, NJ, 1966.
- 37 Kent E Morrison. The fedex problem. *The College Mathematics Journal*, 41(3):222–232, 2010.
- 38 Gyula Sz Nagy. Tschirnhaus’sche Eiflächen und Eikurven. *Acta Mathematica Academiae Scientiarum Hungarica*, 1(1):36–45, 1950.
- 39 Nguyen Mau Nam. The Fermat-Torricelli problem in the light of convex analysis. *ArXiv e-prints*, 2013. arXiv:1302.5244v3.
- 40 Karl Nickel. Triplex-algol and applications. *Interner Bericht des Instituts für Informatik der Universität Karlsruhe*, 1969.
- 41 Karl Nickel. On the Newton method in interval analysis. Technical report, Wisconsin University-Madison Mathematics Research Center, 1971.
- 42 Jiawang Nie, Pablo A. Parrilo, and Bernd Sturmfels. Semidefinite representation of the k -ellipse. In *Algorithms in algebraic geometry*, pages 117–132. Springer, 2008.
- 43 Lawrence M. Ostresh Jr. Convergence and descent in the Fermat location problem. *Transportation Science*, 12(2):153–164, 1978.
- 44 Evanthia Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.
- 45 Pablo A. Parrilo and Bernd Sturmfels. Minimizing polynomial functions. *Algorithmic and quantitative real algebraic geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 60:83–99, 2003.
- 46 Maja Petrović, Bojan Banjac, and Branko Malešević. The geometry of trifocal curves with applications in architecture, urban and spatial planning. *Spatium*, pages 28–33, 2014.
- 47 Simon Plantinga and Gert Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 245–254. ACM, 2004.
- 48 Helmut Ratschek and Jon Rokne. *Computer methods for the range of functions*. Horwood, 1984.
- 49 Gerhard Reinelt. TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.
- 50 Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- 51 Junpei Sekino. n -ellipses and the minimum distance sum problem. *The American mathematical monthly*, 106(3):193–202, 1999.

- 52 Sergey P Shary. Krawczyk operator revised. *Novosibirsk, Institute of Computational Technologies, Russia*, 2004.
- 53 Rudolf Sturm. Über den Punkt kleinster Entfernungssumme von gegebenen Punkten. *Journal für die reine und angewandte Mathematik*, 97:49–61, 1884.
- 54 Warwick Tucker. *Validated Numerics: A short intro to rigorous computations*. Princeton Press, 2011.
- 55 Ehrenfried Walther von Tschirnhaus. *Medicina Mentis Et Corporis*. Fritsch, Lipsiae, 1695. URL: <http://mdz-nbn-resolving.de/urn:nbn:de:bvb:12-bsb10008248-3>.
- 56 Cong Wang, Yi-Jen Chiang, and Chee Yap. On soft predicates in subdivision motion planning. *Computational Geometry: Theory and Applications.*, 48(8):589–605, 2015.
- 57 Alfred Weber. Über den Standort der Industrien. *English translation by CJ Friedrich (1929) Theory of the Location of Industries*, 1909.
- 58 Endre Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Mathematical Journal, First Series*, 43:355–386, 1937.
- 59 Juan Xu and Chee Yap. Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis. In *Proc. International Symposium on Symbolic and Algebraic Computation*, pages 399–406. ACM, 2019.
- 60 Guoliang Xue and Yinyu Ye. An efficient algorithm for minimizing a sum of Euclidean norms with applications. *SIAM Journal on Optimization*, 7(4):1017–1036, 1997.

Parameterized Algorithms for Diverse Multistage Problems

Leon Kellerhals  

Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Malte Renken  

Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Philipp Zschoche  

Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Abstract

The world is rarely static – many problems need not only be solved once but repeatedly, under changing conditions. This setting is addressed by the *multistage* view on computational problems. We study the *diverse multistage* variant, where consecutive solutions of large variety are preferable to similar ones, e.g. for reasons of fairness or wear minimization. While some aspects of this model have been tackled before, we introduce a framework allowing us to prove that a number of diverse multistage problems are fixed-parameter tractable by diversity, namely PERFECT MATCHING, s - t PATH, MATROID INDEPENDENT SET, and PLURALITY VOTING. This is achieved by first solving special, colored variants of these problems, which might also be of independent interest.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Temporal graphs, dissimilar solutions, fixed-parameter tractability, perfect matchings, s - t paths, committee election, spanning forests, matroids

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.55

Related Version *Full Version*: <https://arxiv.org/abs/2105.04856>

Funding *Malte Renken*: Supported by the German Research Foundation (DFG), project MATE (NI 369/17).

Acknowledgements The authors wish to thank Rolf Niedermeier and anonymous reviewers for their careful reading and suggestions of the manuscript. This work was initiated at the research retreat of the Algorithmics and Computational Complexity group of TU Berlin in September 2020 in Zinnowitz.

1 Introduction

In the *multistage* setting, given a sequence of instances of some problem, one asks whether there is a corresponding sequence of solutions such that consecutive solutions relate in some way to each other. Often the aim is to find consecutive solutions that are very *similar* [25, 18, 20, 7, 6, 19]. This is reasonable when changing between distinct solutions incurs some form of cost. In other settings, the opposite goal is more reasonable, that is, consecutive solutions should be very *different*. This is a natural goal when wear minimization, load distribution, or resilience against failures or attacks are of interest. This *diverse multistage* setting is what we want to focus on in this paper. Here, given a sequence of instances of some decision problem, the task is to find a sequence of solutions such that the *diversity*, i.e., the size of the symmetric difference of any two consecutive solutions is *at least* ℓ .



© Leon Kellerhals, Malte Renken, and Philipp Zschoche;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 55; pp. 55:1–55:17

Leibniz International Proceedings in Informatics



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

This problem has already received some attention in the literature: Fluschnik et al. [21] studied the problem of finding diverse s - t paths and Bredebeck et al. [11] considered series of committee elections. In a similar setting, but aiming for large symmetric difference between every two (i.e., not just consecutive) solutions, Baste et al. [8] provide a framework for parameterization by treewidth, while the case that all instances are the same is studied for matchings, independent sets of matroids [22, 23], and for Kemeny rank aggregation [3].

We briefly give a formal definition. Assume Π to be some decision problem which asks whether the *family of solutions* $\mathcal{R}(I) \subseteq 2^{\mathcal{B}(I)}$ of an instance I of Π is non-empty, where $\mathcal{B}(I)$ is some *base set* encompassing all possible solutions. For example, for an instance I of VERTEX COVER, the set $\mathcal{B}(I)$ is the set of all vertices and $\mathcal{R}(I)$ is the set of all vertex covers within the size bound. The problem DIVERSE MULTISTAGE Π is now the following.

DIVERSE MULTISTAGE Π

Input: A sequence $(I_i)_{i=1}^\tau$ of instances of Π and an integer $\ell \in \mathbb{N}_0$.

Question: Is there a sequence $(S_i)_{i=1}^\tau$ of solutions $S_i \in \mathcal{R}(I_i)$ such that $|S_i \Delta S_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$?

Our contributions. We present a general framework which allows us to prove fixed-parameter tractability of DIVERSE MULTISTAGE Π parameterized by the diversity ℓ for several problems Π . This includes finding diverse matchings, but also diverse committees (answering an open question by Bredebeck et al. [11]), diverse s - t paths, and diverse independent sets in matroids such as spanning forests. Finally, we show that similar results cannot be expected for finding diverse vertex covers.

Generally, our framework can be applied to DIVERSE MULTISTAGE Π whenever one can solve a 4-colored variant of Π efficiently. Formally, this variant is defined as follows.

4-COLORED EXACT Π

Input: An instance I of Π , a coloring $c: \mathcal{B}(I) \rightarrow [4]$, and $n_i \in \mathbb{N}_0, i \in [4]$.

Output: A solution $S \in \mathcal{R}(I)$ such that $|\{x \in S \mid c(x) = i\}| = n_i$ for all $i \in [4]$ or “no” if no such solution exists.

Our main result reads as follows.

► **Theorem 1.** *For any parameter r of Π , if an instance I of 4-COLORED EXACT Π can be solved in $f(r) \cdot |I|^{O(1)}$ time, then an instance J of DIVERSE MULTISTAGE Π can be solved in $2^{O(\ell)} \cdot f(r_{\max}) \cdot |J|^{O(1)}$ time, where r_{\max} is the maximum of parameter r over all instances of Π in J .¹*

We prove Theorem 1 in Section 3 in a more general form which also allows solving 4-COLORED EXACT Π by a Monte Carlo algorithm. We then apply our framework to the following problems:

Committee Election (Section 4). In DIVERSE MULTISTAGE PLURALITY VOTING, we are given a set A of agents, a set C of candidates, and τ many *voting profiles* $u_i: A \rightarrow C$. The goal is to find a sequence $(C_i)_{i=1}^\tau$ of committees $C_i \subseteq C$ such that each committee C_i is of size at most k and gets at least x votes in the voting profile u_i (i.e., $|u_i^{-1}(C_i)| \geq x$), and $|C_i \Delta C_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$. We show that there is a $2^{O(\ell)} \cdot |J|^{O(1)}$ -time algorithm to solve a DIVERSE MULTISTAGE PLURALITY VOTING instance J . This answers an open question of Bredebeck et al. [11]. In the full version we generalize the algorithm used to solve 4-COLORED EXACT PLURALITY VOTING to matroids.

¹ For example, if the input is a sequence of graphs and r is the treewidth, then r_{\max} is the maximum treewidth over all graphs in the input.

Perfect Matching (Section 5). In the multistage setting, Perfect Matching is among the problems most intensively studied [25, 5, 4, 13, 34]. Given a sequence of graphs $(G_i)_{i=1}^\tau$ and an integer ℓ , DIVERSE MULTISTAGE PERFECT MATCHING asks whether there is a sequence $(M_i)_{i=1}^\tau$ such that each M_i is a perfect matching in G_i , and $|M_i \Delta M_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$. We show that there is a randomized $2^{O(\ell)} \cdot |J|^{O(1)}$ -time algorithm to solve a DIVERSE MULTISTAGE PERFECT MATCHING instance J with constant error probability. This stands in remarkable contrast to the $W[1]$ -hardness of the (non-diverse) MULTISTAGE PERFECT MATCHING, when parameterized by $\ell + \tau$ [34]. To apply our framework, we establish an algebraic algorithm using the Pfaffian of a specific variant of the Tutte matrix to solve s -COLORED EXACT PERFECT MATCHING on an n -vertex graph in $n^{O(s)}$ time with low error probability.

s-t Path (Section 6). Studying s - t PATH in the multistage setting was already suggested in the seminal work of Gupta et al. [25]. In DIVERSE MULTISTAGE s - t PATH one is given a sequence of graphs $(G_i)_{i=1}^\tau$, two distinct vertices s and t , and an integer ℓ , and asks whether there is a sequence $(P_i)_{i=1}^\tau$ such that each P_i is an s - t PATH in G_i , and $|V(P_i) \Delta V(P_{i+1})| \geq \ell$ for all $i \in [\tau - 1]$. Fluschnik et al. [21] provided a comprehensive study of finding s - t paths of bounded length in the multistage setting from the viewpoint of parameterized complexity. Among other results, they showed that DIVERSE MULTISTAGE s - t PATH is NP-hard but fixed-parameter tractable when parameterized by the maximum length of an s - t PATH in the solution. We show that DIVERSE MULTISTAGE s - t PATH parameterized by ℓ is fixed-parameter tractable. At first glance, using our framework seems unpromising since 4-COLORED EXACT s - t PATH can presumably not be solved in polynomial time (it is NP-hard by a straight-forward reduction from HAMILTONIAN PATH). However, we develop a win/win strategy around a generalization of the Erdős-Pósa theorem for long cycles due to Mousset et al. [30] so that we have to solve 4-COLORED EXACT s - t PATH only on graphs on which the treewidth is upper-bounded in the parameter ℓ .

In Section 7, we complement our fixed-parameter tractability results with a $W[1]$ -hardness for DIVERSE MULTISTAGE VERTEX COVER when parameterized by ℓ .

2 Preliminaries

We denote by \mathbb{N} and \mathbb{N}_0 the natural numbers excluding and including zero, respectively. For $n \in \mathbb{N}$, let $[n] := \{1, 2, \dots, n\}$. For two sets A and B , we denote by $A \Delta B := (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$ the *symmetric difference* of A and B , and by $A \uplus B$ the disjoint union of A and B . For a function $c: A \rightarrow B$, let $c(A') := \bigcup_{a \in A'} c(a)$ and $c^{-1}(b) := \{a \in A \mid c(a) = b\}$, where $A' \subseteq A$. We also use the notations c^b and $c^{b, b'}$ as shorthands for $c^{-1}(b)$ and $c^{-1}(b) \cup c^{-1}(b')$, respectively.

A *Monte Carlo algorithm*, or an algorithm with error probability p , is a randomized algorithm that returns a correct answer with probability $1 - p$.

Let Σ be a finite alphabet. A *parameterized problem* L is a subset $L \subseteq \{(x, k) \in \Sigma^* \times \mathbb{N}_0\}$. An instance $(x, k) \in \Sigma^* \times \mathbb{N}_0$ is a *yes-instance* of L if and only if $(x, k) \in L$ (otherwise, it is a *no-instance*). A parameterized problem L is *fixed-parameter tractable* (in FPT) if for every input (x, k) one can decide in $f(k) \cdot |x|^{O(1)}$ time whether $(x, k) \in L$, where f is some computable function only depending on k . If $W[1]$ -hard parameterized problem is not fixed-parameter tractable unless $FPT = W[1]$. We refer to Downey and Fellows [17] and Cygan et al. [14] for more material on parameterized complexity. We use standard notation from graph theory [16]. Throughout this paper, we assume graphs to be simple and undirected.

3 The General Framework

In this section, we introduce a general framework to show (for some decision problem Π) fixed-parameter tractability of DIVERSE MULTISTAGE Π parameterized by ℓ . Recall that, for every instance I of a decision problem Π , we denote by $\mathcal{B}(I)$ the base set encompassing all possible solutions, by $\mathcal{R}(I) \subseteq 2^{\mathcal{B}(I)}$ the family of solutions, and by $|I|$ the input size of I , which is at least $|\mathcal{B}(I)|$. For the remainder of this section we assume that $|\mathcal{B}(I)| \geq 2$ for all instances I of Π . The framework is applicable to DIVERSE MULTISTAGE Π if there is an efficient algorithm for 4-COLORED EXACT Π . Formally, we use the following prerequisite, which is slightly more general than in Theorem 1.

► **Assumption 2.** There are computable functions f, g such that for every $0 \leq p \leq 1$ for which $g(p)$ is defined, there is a Monte-Carlo algorithm \mathcal{A} with error probability p and running time $f(r) \cdot |I|^{O(1)} \cdot g(p)$ that solves an instance I of 4-COLORED EXACT Π , where $r \in \mathbb{N}_0$ is some parameter of I and g is monotone non-increasing.

We allow an error probability in Assumption 2 because for one of our applications (in Section 5), no other polynomial-time algorithm is known. The goal is to prove the following.

► **Theorem 3.** *Let Assumption 2 be true. Then any size- n instance I of DIVERSE MULTISTAGE Π can be solved in $2^{O(\ell)} \cdot f(r_{\max}) \cdot n^{O(1)} \cdot g(p/\tau 2^{O(\ell)} n^{O(1)})$ time by a Monte-Carlo algorithm with error probability p , where r_{\max} is the maximum of parameter r over all instances of Π in I , and $0 \leq p \leq 1$ is an arbitrary probability for which the above expression is defined.²*

Note that, if we have a non-randomized algorithm in Assumption 2 (that is, $g(0)$ is defined and g maps always to one), then Theorem 1 follows directly from Theorem 3.

The underlying strategy of the algorithm for a DIVERSE MULTISTAGE Π -instance J behind Theorem 3 is to compute for each instance I of Π in J a solution family such that the Cartesian product of these families contains a solution for J if and only if J is a *yes*-instance. Once these families are obtained, we can check whether J is a *yes*-instance by dynamic programming. To this end, we compute a small subset of $\mathcal{R}(I)$ satisfying the following definition.

► **Definition 4.** *Let \mathcal{F} be a set family. A subfamily $\hat{\mathcal{F}} \subseteq \mathcal{F}$ is called an ℓ -diverse representative of \mathcal{F} if the following holds: for any pair of sets A, B , if there is an $S \in \mathcal{F}$ with $\min\{|A \Delta S|, |B \Delta S|\} \geq \ell$, then there is an $\hat{S} \in \hat{\mathcal{F}}$ such that $\min\{|A \Delta \hat{S}|, |B \Delta \hat{S}|\} \geq \ell$.*

First of all, we note that ℓ -diverse representatives can be rather small.

► **Lemma 5.** *Let \mathcal{F} be a set family and $S_1, S_2, S_3 \in \mathcal{F}$. If $|S_i \Delta S_j| \geq 2\ell$ for all distinct $i, j \in [3]$, then $\{S_1, S_2, S_3\}$ is an ℓ -diverse representative of \mathcal{F} .*

Proof. Assume for contradiction that there exist sets A and B with $\min\{|A \Delta S_i|, |B \Delta S_i|\} < \ell$ for all i . Without loss of generality, assume that $|A \Delta S_1| < \ell$. Then for $j \in \{2, 3\}$ we have $|A \Delta S_j| \geq |S_1 \Delta S_j| - |S_1 \Delta A| > 2\ell - \ell = \ell$ by the triangle inequality. Therefore, $|B \Delta S_j| < \ell$ for all $j \in \{2, 3\}$. In particular, $|B \Delta S_2| < \ell$. Again, by the triangle inequality $|B \Delta S_3| \geq |S_2 \Delta S_3| - |S_2 \Delta B| > 2\ell - \ell = \ell$, i.e., $\min\{|A \Delta S_3|, |B \Delta S_3|\} \geq \ell$ – a contradiction. ◀

² For example, if we only have an algorithm with non-zero error probability, then $p = 0$ is excluded.

In the following, we measure the distance of two solutions by the size of the symmetric difference. In a nutshell, we compute an ℓ -diverse representative of the family of solutions by first trying to compute three solutions which are far apart from each other (that is, size of symmetric difference at least 2ℓ). If this succeeds, then by Lemma 5 we are done. Otherwise, we distinguish between three cases.

No solution. If there is no solution at all, then trivially \emptyset is an ℓ -diverse representative of the family of solutions.

One solution. If we only find one solution S_1 to the instance of Π , then each other solution is close to S_1 . Hence, for any two sets A, B , if one of them is far away from S_1 , then by the triangle inequality it is also far away from every other solution and can be safely ignored. For those sets which are close to S_1 , we can exploit the upper bound on the symmetric difference by using color-coding [2] and then applying Assumption 2 to compute an ℓ -diverse representative of the family of solutions. This case is handled in Lemma 9.

Two solutions. If we find two diverse solutions S_1 and S_2 such that no other solution is far away from both, then S_1 and S_2 partition the solution space into two parts: the solutions close to S_1 and those close to S_2 . Again, given two sets A, B , if either of them is far away from S_1 and S_2 , then we may ignore it. By including S_1 and S_2 in our family, we may further assume that A is similar to S_1 and B is similar to S_2 . We distinguish two subcases. If the distance between S_1 and S_2 is very large, then A is far away from all solutions in the second part and B is far away from all solutions in the first part. We can thus ignore one of them (say B) and exploit the fact that A, S_1 , and all solutions of interest are close to each other to use color-coding and then apply Assumption 2. In the other subcase where the distance between S_1 and S_2 is bounded, we can utilize that fact similarly. This case is handled in Lemma 10.

Hereafter, the details. Before we dive into the case distinction outlined above, we need to prove two technical lemmata, telling us how to build a diverse representative set that works for all sets obeying some given coloring of the elements of $\mathcal{B}(I)$. These will later work as building blocks in the construction of proper diverse representatives. In the first lemma, only two colors are used, and we are only concerned with one arbitrary set A instead of two.

► **Lemma 6.** *Let Assumption 2 be true. Given an instance I of Π of size n , a coloring $c: \mathcal{B}(I) \rightarrow [2]$, and a solution $M \in \mathcal{R}(I)$, one can compute in $f(r)n^{O(1)}g(pn^{-4})$ time and with error probability at most p a family $\mathcal{F} \subseteq \mathcal{R}(I)$ of size at most n^4 such that for any $S \in \mathcal{R}(I)$ and any $A \subseteq \mathcal{B}(I)$ with $S \setminus A \subseteq c^1$ and $A \setminus S \subseteq c^2$, there is $\hat{S} \in \mathcal{F}$ with $|A\Delta\hat{S}| \geq |A\Delta S|$ and $|M\Delta\hat{S}| = |M\Delta S|$.*

Proof. Let $F'_1 := c^1 \cap M$, $F'_2 := c^2 \cap M$, $F'_3 := c^1 \setminus M$, and $F'_4 := c^2 \setminus M$.

Start with $\mathcal{F} = \emptyset$. Then, for each $m \in [n]$ and each number partition $\sum_{i=1}^4 m_i = m$ with $m_i \geq 0$, use algorithm \mathcal{A} to search in $f(r)n^{O(1)}g(pn^{-4})$ time and with error probability at most pn^{-4} for a set $N \in \mathcal{R}(I)$ such that $|N \cap F'_i| = m_i$ for all $i \in [4]$. If this succeeds, then we add N to \mathcal{F} . Since there are $\binom{n+4}{4} \leq n^4$ possibilities for m_1, \dots, m_4 , the probability of an error occurring is upper-bounded by p . Moreover, the size of \mathcal{F} is upper-bounded by n^4 and hence the time required is bounded by $f(r)n^{O(1)}g(pn^{-4})$.

It remains to be proven that \mathcal{F} has the desired properties. Let $S \in \mathcal{R}(I)$ be arbitrary and set $m_i := |S \cap F'_i|$ for all $i \in [4]$. By construction, \mathcal{F} contains a set $\hat{S} \in \mathcal{R}(I)$ such that $|\hat{S} \cap F'_i| = m_i$. We then have $|\hat{S}\Delta M| = m_3 + m_4 + |M| - m_1 - m_2 = |S\Delta M|$.

Let $A \subseteq \mathcal{B}(I)$ be a set with $S \setminus A \subseteq c^1$ and $A \setminus S \subseteq c^2$. Since $A \setminus S \subseteq c^2$ we have

$$|A \cap S \cap c^1| = |A \cap c^1| \geq |A \cap \hat{S} \cap c^1| \quad (1)$$

and since $S \setminus A \subseteq c^1$, we have that

$$|A \cap S \cap c^2| = |S \cap c^2| = m_2 + m_4 = |\hat{S} \cap c^2| \geq |A \cap \hat{S} \cap c^2|. \quad (2)$$

By adding (1) and (2) we obtain $|A \cap S| \geq |A \cap \hat{S}|$ which in turn implies $|A \Delta S| \leq |A \Delta \hat{S}|$ since $|S| = |\hat{S}|$. \blacktriangleleft

The next lemma extends the approach of Lemma 6 to the case where we have four colors and two arbitrary sets A, B .

► Lemma 7. *Let Assumption 2 be true. Given an instance I of Π of size n and a coloring $c: \mathcal{B}(I) \rightarrow [4]$, one can compute in $f(r)n^{O(1)}g(pn^{-4})$ time and with error probability at most p a family $\mathcal{F} \subseteq \mathcal{R}(I)$ of size at most n^4 such that for any $S \in \mathcal{R}(I)$ and all sets $A, B \subseteq \mathcal{B}(I)$ with $A \setminus (B \cup S) \subseteq c^1$, $B \setminus (A \cup S) \subseteq c^2$, $(A \cap B) \setminus S \subseteq c^3$, and $S \setminus (A \cap B) \subseteq c^4$, there is $\hat{S} \in \mathcal{F}$ with $|C \Delta \hat{S}| \geq |C \Delta S|$ for all $C \in \{A, B\}$.*

Proof. Begin with $\mathcal{F} = \emptyset$. Then, for each $m \in [n]$ and each number partition $\sum_{i=1}^4 m_i = m$ with $m_i \geq 0$, use algorithm \mathcal{A} to search in $f(r)n^{O(1)}g(pn^{-4})$ time and with error probability at most pn^{-4} for an $M \in \mathcal{R}(I)$ such that $|M \cap c^i| = m_i$ for all $i \in [4]$. If this succeeds, then add M to \mathcal{F} . Since there are $\binom{n+4}{4} \leq n^4$ possibilities for m_1, \dots, m_4 , the probability of an error occurring is upper-bounded by p . Moreover, the size of \mathcal{F} is at most n^4 and thus the overall running time is $f(r)n^{O(1)}g(pn^{-4})$.

Now let $S \in \mathcal{R}(I)$ be arbitrary. Set $m_i := |S \cap c^i|$, for all $i \in [4]$. By construction there is $\hat{S} \in \mathcal{F}$ such that $|\hat{S} \cap c^i| = m_i$ for all $i \in [4]$. It remains to be proven that \hat{S} has the desired properties. To this end, let $A, B \subseteq \mathcal{B}(I)$ be two sets as stated in the lemma. By symmetry, it suffices to show that $|A \Delta \hat{S}| \geq |A \Delta S|$.

Since $S \setminus A \subseteq c^4$ we have

$$|S \cap A \cap c^{1,3}| = |S \cap c^{1,3}| = m_1 + m_3 = |\hat{S} \cap c^{1,3}| \geq |\hat{S} \cap A \cap c^{1,3}| \quad (3)$$

and since $A \setminus S \subseteq c^{1,3}$, we have

$$|S \cap A \cap c^{2,4}| = |A \cap c^{2,4}| \geq |\hat{S} \cap A \cap c^{2,4}|. \quad (4)$$

By adding (3) and (4), we obtain $|S \cap A| \geq |\hat{S} \cap A|$ and thus $|S \Delta A| \leq |\hat{S} \Delta A|$ since $|S| = |\hat{S}|$. \blacktriangleleft

We now describe how we generate the colorings required for using Lemmata 6 and 7. Color-coding [2] is well-established in the toolbox of parameterized algorithms. While color-coding was initially described as a randomized technique, we use universal sets [33] to derandomize this technique as shown in the next lemma. Interestingly, without this derandomization the error probability of the color-coding step would later propagate through the dynamic programming method and consequently also depend on the number of instances of Π in the input instance of DIVERSE MULTISTAGE Π . The derandomization works as follows.

► Lemma 8. *For any set A of size n and any $b \leq n$ one can compute in $2^{2b+o(b)} \log n \cdot n$ time a family of functions $\{c_j: A \rightarrow [4] \mid j \in [2^{2b+o(b)} \log n]\}$ such that for any $\bigsqcup_{i=1}^4 B_i \subseteq A$ with $|\bigsqcup_{i=1}^4 B_i| \leq b$ there is a j such that $c_j(B_i) = \{i\}$, for all $i \in [4]$.*

Proof. Let $A := \{a_1, \dots, a_n\}$. By a result of Naor et al. [citeNearOptimalDerandomization], one can compute in $2^{2b} b^{O(\log b)} \cdot \log n \cdot n \subseteq 2^{2b+o(b)} \log n \cdot n$ time a so-called $(2n, 2b)$ -universal set which is a family $\mathcal{U} \subseteq 2^{[2n]}$ such that for every $B' \subseteq A$ with $|B'| = 2b$ the family $\{B' \cap U \mid U \in \mathcal{U}\}$ contains all 2^{2b} subsets of B' . Let $\mathcal{U} := \{U_i\}_{i=1}^{2^{2b+o(b)} \log n}$. We then define

$c_j, j \in [2^{2b+o(b)} \log n]$, by

$$c_j(a_i) := \begin{cases} 1, & \text{if } i, i+n \in U_j, \\ 2, & \text{if } i \in U_j \text{ and } i+n \notin U_j, \\ 3, & \text{if } i \notin U_j \text{ and } i+n \in U_j, \text{ and} \\ 4, & \text{if } i, i+n \notin U_j. \end{cases}$$

Now let $B_1 \uplus B_2 \uplus B_3 \uplus B_4 \subseteq A$ be an arbitrary 4-partition of a subset of A of size at most b . Consider $B' := \{i, i+n \mid a_i \in \bigcup_{q=1}^4 B_q\}$. We assume that B' is of size $2b$, otherwise we add arbitrary elements from $[2n]$. Since $B'' := \{i, i+n \mid a_i \in B_1\} \cup \{i \mid a_i \in B_2\} \cup \{i+n \mid a_i \in B_3\} \subseteq B'$ there is an $U_j \in \mathcal{U}$ such that $B' \cap U_j = B''$. Hence, $c_j(B_i) = \{i\}$, for all $i \in [4]$. ◀

We now show how to generate an ℓ -diverse representative of the family of solutions if there is one solution M^* from which no other solution differs by more than 2ℓ .

► **Lemma 9.** *Let Assumption 2 be true. Given an instance I of Π of size n , and a solution $M^* \in \mathcal{R}(I)$ such that each $M \in \mathcal{R}(I)$ satisfies $|M \Delta M^*| \leq 2\ell$, one can compute in $2^{16\ell+o(\ell)} \log n \cdot f(r) \cdot n^{O(1)} \cdot g(p/2^{16\ell+o(\ell)} \log n \cdot n^4)$ time and with error probability p an ℓ -diverse representative of $\mathcal{R}(I)$ of size at most $2^{16\ell+o(\ell)} \log n \cdot n^4$.*

Proof. For simplicity, let $\mathbb{J} := [2^{16\ell+o(\ell)} \log n]$. Apply Lemma 8 with $b = 8\ell$ to compute in $2^{16\ell+o(\ell)} \log n \cdot n$ time a family of colorings $\{c_j: \mathcal{B}(I) \rightarrow [4] \mid j \in \mathbb{J}\}$. To apply Lemma 8 we assume that $|\mathcal{B}(I)| \geq 8\ell$, otherwise we utilize dummy elements. For each $j \in \mathbb{J}$, apply Lemma 7 to I and c_j to compute a family $\mathcal{F}_j \subseteq \mathcal{R}(I)$ with error probability $p \cdot |\mathbb{J}|^{-1}$. Observe that the probability of an error occurring at any of the $|\mathbb{J}|$ steps is bounded by p . Choose $\mathcal{F} := \{M^*\} \cup \bigcup_{j \in \mathbb{J}} \mathcal{F}_j$. According to Lemma 7 the size of \mathcal{F} is upper-bounded by $|\mathbb{J}| \cdot n^4$ and the time required is bounded by $|\mathbb{J}| \cdot f(r) \cdot n^{O(1)} \cdot g(pn^{-4} \cdot |\mathbb{J}|^{-1})$.

We now show that \mathcal{F} is an ℓ -diverse representative of $\mathcal{R}(I)$. To this end, let $S \in \mathcal{R}(I)$ and let A, B be two arbitrary sets such that $|A \Delta S| \geq \ell$ and $|B \Delta S| \geq \ell$. Since $M^* \in \mathcal{F}$, we may assume by symmetry that, say, $|M^* \Delta A| < \ell$, otherwise we are done. Note that $|M^* \Delta S| \leq 2\ell$ and that $|A \Delta S| \leq |A \Delta M^*| + |M^* \Delta S| < 3\ell$. We say that some coloring c is *good* for A, B, S if the conditions of Lemma 7 are satisfied, i.e. if

$$A \setminus (B \cup S) \subseteq c^1, \quad B \setminus (A \cup S) \subseteq c^2, \quad (A \cap B) \setminus S \subseteq c^3, \quad \text{and} \quad S \setminus (A \cap B) \subseteq c^4.$$

We distinguish between two cases.

Case 1: $|M^* \Delta B| < 3\ell$. Then $|B \Delta S| \leq |B \Delta M^*| + |M^* \Delta S| \leq 5\ell$. According to Lemma 8 there is an $i \in \mathbb{J}$ such that coloring c_i is good for A, B, S , since $|B \Delta S| + |A \Delta S| < 8\ell$. By Lemma 7 and construction of \mathcal{F}_i , there is an $\hat{S} \in \mathcal{F}_i \subseteq \mathcal{F}$ such that $|\hat{S} \Delta A| \geq |S \Delta A| \geq \ell$ and $|\hat{S} \Delta B| \geq |S \Delta B| \geq \ell$.

Case 2: $|M^* \Delta B| \geq 3\ell$. Set $B' := A$. According to Lemma 8 there is an $i \in \mathbb{J}$ such that coloring c_i is good for A, B', S , since $|B' \Delta S| + |A \Delta S| < 6\ell$. Thus, by Lemma 7 and by the construction of \mathcal{F}_i there is an $\hat{S} \in \mathcal{F}_i \subseteq \mathcal{F}$ such that $|\hat{S} \Delta A| \geq |S \Delta A| \geq \ell$. Finally, we observe that $|\hat{S} \Delta B| \geq |M^* \Delta B| - |M^* \Delta \hat{S}| \geq 3\ell - 2\ell \geq \ell$ by the triangle inequality.

This completes the proof. ◀

Next, we show how to generate an ℓ -diverse representative of the family of solutions if there are two solutions such that no other solution differs from both by more than 2ℓ .

► **Lemma 10.** *Let Assumption 2 be true. Let I be an Π -instance of size n , and $M_1, M_2 \in \mathcal{R}(I)$ such that $|M_1 \Delta M_2| \geq 2\ell$ and each $M \in \mathcal{R}(I)$ has $\min\{|M \Delta M_1|, |M \Delta M_2|\} \leq 2\ell$. Then one can compute, in $2^{20\ell+o(\ell)} \log n \cdot f(r) n^{O(1)} g(p/n^{420\ell+o(\ell)} \log n)$ time and with error probability p , an ℓ -diverse representative of $\mathcal{R}(I)$ of size $2^{20\ell+o(\ell)} \log n \cdot n^4$.*

Proof. For simplicity, let $\mathbb{J} := \lceil 2^{20\ell+o(\ell)} \log n \rceil$. Apply Lemma 8 with $b = 10\ell$ to compute in $2^{20\ell+o(\ell)} \log n \cdot n$ time a family of colorings $\{c_j: \mathcal{B}(I) \rightarrow [4] \mid j \in \mathbb{J}\}$. To apply Lemma 8 we assume that $|\mathcal{B}(I)| \geq 10\ell$, otherwise we utilize dummy elements.

For each $j \in \mathbb{J}$, apply Lemma 7 to I and c_j to compute a family $\mathcal{F}_j \subseteq \mathcal{R}(I)$ of size at most n^4 with error probability $p/3 \cdot |\mathbb{J}|^{-1}$. Observe that the probability of an error occurring at any of the $|\mathbb{J}|$ steps is upper-bounded by $p/3$ and the computation of all \mathcal{F}_j takes $|\mathbb{J}|f(r)n^{\mathcal{O}(1)}g(p/3n^4 \cdot |\mathbb{J}|)$ time.

Next, define another family of colorings $\{c'_j: \mathcal{B}(I) \rightarrow [2] \mid j \in \mathbb{J}\}$ by setting $c'_j(x) := \lceil c_j(x)/2 \rceil$. Then, for each $j \in \mathbb{J}$, apply Lemma 6, to I , c'_j and M_1 to compute a family $\mathcal{F}'_j \subseteq \mathcal{R}(I)$, with the same error probability and time bound as before. Repeat with M_2 instead of M_1 to obtain \mathcal{F}''_j .

Set $\mathcal{F} := \{M_1, M_2\} \cup \bigcup_{j \in \mathbb{J}} (\mathcal{F}_j \cup \mathcal{F}'_j \cup \mathcal{F}''_j)$. Then \mathcal{F} has size at most $3|\mathbb{J}|n^4 + 2 \subseteq 2^{20\ell+o(\ell)} \log n \cdot n^4$. Computing \mathcal{F} takes $2^{20\ell+o(\ell)} \log n \cdot f(r)n^{\mathcal{O}(1)}g(p/3n^4 \cdot |\mathbb{J}|)$ time. The probability of an error occurring at any step while computing \mathcal{F} is upper-bounded by p .

We now show that \mathcal{F} is an ℓ -diverse representative of $\mathcal{R}(I)$. To this end, let $S \in \mathcal{R}(I)$ and A, B be two arbitrary sets such that $|A\Delta S| \geq \ell$ and $|B\Delta S| \geq \ell$. We may assume for each $i \in [2]$ that $|M_i\Delta A| < \ell$ or $|M_i\Delta B| < \ell$, otherwise we are done. By symmetry, we may assume $|M_1\Delta A| < \ell$. Then $|M_2\Delta A| \geq |M_2\Delta M_1| - |M_1\Delta A| > \ell$ by the triangle inequality and thus we must have $|M_2\Delta B| < \ell$. By assumption, $\min\{|S\Delta M_1|, |S\Delta M_2|\} \leq 2\ell$, so let without loss of generality $|S\Delta M_1| \leq 2\ell$. Note that $|A\Delta S| \leq |A\Delta M_1| + |M_1\Delta S| < 3\ell$. We distinguish the following two cases.

Case 1: $|M_1\Delta M_2| \leq 4\ell$. Then, $|B\Delta S| \leq |B\Delta M_2| + |M_2\Delta M_1| + |M_1\Delta S| < 7\ell$. We say that some coloring c is *good* for A, B, S if the conditions of Lemma 7 are satisfied, i.e. if $A \setminus (B \cup S) \subseteq c^1$, $B \setminus (A \cup S) \subseteq c^2$, $(A \cap B) \setminus S \subseteq c^3$, and $S \setminus (A \cap B) \subseteq c^4$. According to Lemma 8 there is an $i \in \mathbb{J}$ such that coloring c_i is good for A, B, S , since $|B\Delta S| + |A\Delta S| \leq 10\ell$. By Lemma 7, there is $\hat{S} \in \mathcal{F}_i \subseteq \mathcal{F}$ such that $|\hat{S}\Delta A| \geq |S\Delta A| \geq \ell$ and $|\hat{S}\Delta B| \geq |S\Delta B| \geq \ell$.

Case 2: $|M_1\Delta M_2| > 4\ell$. Since $|S\Delta A| < 3\ell \leq 10\ell$, there is $j \in \mathbb{J}$ such that $S \setminus A \subseteq c_j^1$ and $A \setminus S \subseteq c_j^2$. By Lemma 6 there is $\hat{S} \in \mathcal{F}'_j$ such that $|\hat{S}\Delta M_1| = |S\Delta M_1| \leq 2\ell$ and $|\hat{S}\Delta A| \geq |S\Delta A| \geq \ell$. Finally, observe that by the triangle inequality $|\hat{S}\Delta B| \geq |M_1\Delta M_2| - |M_1\Delta \hat{S}| - |B\Delta M_2| > \ell$.

This completes the proof. \blacktriangleleft

With Lemmata 5, 9, and 10 at hand we can formalize the case distinction outlined in the beginning of the section. This gives us a way to efficiently compute an ℓ -diverse representative in general.

► Lemma 11. *Let Assumption 2 be true. Let I be an instance of Π of size n . One can compute an ℓ -diverse representative of $\mathcal{R}(I)$ of size $2^{20\ell+o(\ell)} \log n \cdot n^4$ in $2^{20\ell+o(\ell)} \log n \cdot f(r)n^{\mathcal{O}(1)}g(p/n^4 \cdot 2^{20\ell+o(\ell)} \log n)$ time with error probability at most p .*

Proof. Our procedure to compute an ℓ -diverse representative of $\mathcal{R}(I)$ works in four steps.

Step 1. We use \mathcal{A} with a monochrome coloring and error probability $p/4n$ to search for some $M_1 \in \mathcal{R}(I)$ in $f(r)n^{\mathcal{O}(1)}g(p/4n)$ time by guessing the size of $|M_1| \leq n$. Observe that the probability of an error occurring in any of the searches is upper-bounded by $p/4$. If we do not succeed, then output the empty set and we are done. Otherwise, we proceed with the next step.

Step 2. For each pair m_1, m_2 with $m_1 + m_2 \leq n$ and $m_2 + |M_1| - m_1 > 2\ell$, try to compute $M_2 \in \mathcal{R}(I)$ with $|M_2 \cap M_1| = m_1$ and $|M_2 \cap (\mathcal{B}(I) \setminus M_1)| = m_2$ in $f(r)n^{O(1)}g(p/4n^2)$ time and with error probability $p/4n^2$ using \mathcal{A} with a 2-coloring where elements in M_1 are assigned one color and elements in $\mathcal{B}(I) \setminus M_1$ are assigned the second color. If no such M_2 is found for any pair m_1, m_2 , then for every $M \in \mathcal{R}(I)$ the symmetric difference $|M \Delta M_1| \leq 2\ell$. In that case we may apply Lemma 9 with error probability $p/2$ and are done. Observe that the probability of an error occurring at any step until here is upper-bounded by p and the overall running time is $2^{16\ell+o(\ell)} \log n \cdot f(r)n^{O(1)}g(p/n^4 \cdot 2^{16\ell+o(\ell)} \log n)$. If we found such an M_2 , then we proceed with the next step.

Step 3. We have $M_1, M_2 \in \mathcal{R}(I)$ with $|M_1 \Delta M_2| \geq 2\ell$. Define the coloring $c: \mathcal{B}(I) \rightarrow [4]$ by

$$c(v) := \begin{cases} i & \text{if } v \in M_i \setminus M_j \text{ for } \{i, j\} = \{1, 2\}, \\ 3 & \text{if } v \in M_1 \cap M_2, \text{ and} \\ 4 & \text{otherwise.} \end{cases}$$

For all m'_1, m'_2, m'_3, m'_4 with $m'_1 + m'_2 + m'_3 + m'_4 \leq n$ and $m'_2 + m'_4 + |M_1| - m'_1 - m'_3 > 2\ell$ and $m'_1 + m'_4 + |M_2| - m'_2 - m'_3 > 2\ell$, search for a solution $M_3 \in \mathcal{R}(I)$ with $|M_3 \cap c^i| = m'_i$, for all $i \in [4]$, using \mathcal{A} with c and error probability $p/4n^4$. For all these combined, we thus have error probability $p/4$ and need $f(r)n^{O(1)}g(p/4n^4)$ time. If no such M_3 is found for any choice of m'_1, m'_2, m'_3, m'_4 , then any $M \in \mathcal{R}(I)$ must have $\min\{|M \Delta M_1|, |M \Delta M_2|\} \leq 2\ell$. In that case we may apply Lemma 10 with error probability $p/4$ and are done. Observe that the probability of an error occurring at any step until here is upper-bounded by p and the overall running time is $2^{20\ell+o(\ell)} \log n \cdot f(r)n^{O(1)}g(p/n^4 \cdot 2^{20\ell+o(\ell)} \log n)$. In case that we found such an M_3 , we proceed with the next step.

Step 4. We have $M_1, M_2, M_3 \in \mathcal{R}(I)$ such that $|M_i \Delta M_j| \geq 2\ell$ for all distinct $i, j \in [3]$. Hence, by Lemma 5, we can output $\{M_1, M_2, M_3\}$. This completes the proof. \blacktriangleleft

Finally, Lemma 11 allows us to formulate a dynamic program for DIVERSE MULTISTAGE II and prove Theorem 3.

Proof of Theorem 3. Let $J := ((I_i)_{i=1}^\tau, \ell)$ be an instance of DIVERSE MULTISTAGE II, where $n := \max_{i \in [\tau]} |I_i|$. For each $i \in [\tau]$ we apply Lemma 11 to obtain an ℓ -diverse representative \mathcal{F}_i of $\mathcal{R}(I_i)$ that has size at most $2^{20\ell+o(\ell)} \log n \cdot n^4$ in $2^{20\ell+o(\ell)} \log n \cdot f(r)n^{O(1)} \cdot g(p/\tau n^4 \cdot 2^{20\ell+o(\ell)} \log n)$ time with error probability p/τ . Observe that the probability of an error occurring at any step is upper-bounded by p . Now we use the following dynamic program to check whether J is a *yes*-instance.

$$\forall i \in \{2, 3, \dots, \tau\}, S \in \mathcal{F}_i: D[i, S] := \begin{cases} \top & \text{if } \exists \widehat{S} \in \mathcal{F}_{i-1}: D[i-1, \widehat{S}] = \top \text{ and } |S \Delta \widehat{S}| \geq \ell, \\ \perp & \text{otherwise,} \end{cases}$$

where $D[1, \widehat{S}] = \top$ if and only if $\widehat{S} \in \mathcal{F}_1$. We report that J is a *yes*-instance if and only there is an $S \in \mathcal{F}_\tau$ such that $D[\tau, S] = \top$. Note that this takes $(2^{20\ell+o(\ell)} \log n \cdot n^4)^2 \tau \subseteq 2^{O(\ell)} n^{O(1)} \tau$ time. Hence our overall running time is $2^{O(\ell)} \cdot f(r_{\max})n^{O(1)} \cdot \tau \cdot g(p/\tau n^4 \cdot 2^{20\ell+o(\ell)} \log n)$, where r_{\max} is the maximum of parameter r over all instances of II in J .

(\Leftarrow): We show by induction over $i \in [\tau]$ that if $D[i, S] = \top$, then there is a sequence $(S_j)_{j \in [i]}$ such that $S_i = S$, $S_j \in \mathcal{R}(I_j)$ for all $j \in [i]$ and $|S_{j-1} \Delta S_j| \geq \ell$ for all $j \in \{2, 3, \dots, i\}$.

By definition of D this is clearly the case for $i = 1$. Now let $1 < i \leq \tau$ and $D[i, S] = \top$. Since $D[i, S] = \top$, $S \in \mathcal{F}_i$ and thus $S \in \mathcal{R}(I_i)$. By definition of D there is an $\widehat{S} \in \mathcal{F}_{i-1}$ with $D[i-1, \widehat{S}] = \top$ and $|S \Delta \widehat{S}| \geq \ell$. By induction hypothesis, there is a sequence $(S_j)_{j \in [i-1]}$ such

that $S_{i-1} = \widehat{S}$, $S_j \in \mathcal{R}(I_j)$ for all $j \in [i-1]$ and $|S_{j-1} \Delta S_j| \geq \ell$ for all $j \in \{2, 3, \dots, i-1\}$. Hence, the sequence $(S_1, \dots, S_{i-1} = \widehat{S}, S)$ completes the induction. Thus, if we report that J is a *yes*-instance, then this is true.

(\Rightarrow): Now let $(S_j)_{j \in [\tau]}$ be a solution for J . To simplify the proof let $S_{\tau+1}$ be a set of ℓ elements that are disjoint from S_τ . We show by induction that for all $i \in [\tau]$ there is a $Z \in \mathcal{F}_i$ such that $D[i, Z] = \top$ and $|Z \Delta S_{i+1}| \geq \ell$.

Let $i = 1$. Then there is a $Z \in \mathcal{F}_1$ such that $|S_2 \Delta Z| \geq \ell$ since \mathcal{F}_1 is an ℓ -diverse representative of $\mathcal{R}(I_1)$. Hence, $D[1, Z] = \top$.

Now let $1 < i \leq \tau$. By induction hypothesis, there is a $Z_{i-1} \in \mathcal{F}_{i-1}$ such that $D[i-1, Z_{i-1}] = \top$ and $|S_i \Delta Z_{i-1}| \geq \ell$. Since $S_i \in \mathcal{R}(I_i)$ and we have $|S_i \Delta Z_{i-1}|, |S_i \Delta S_{i+1}| \geq \ell$ and \mathcal{F}_i is an ℓ -diverse representative of $\mathcal{R}(I_i)$, there is a $Z \in \mathcal{F}_i$ such that $|Z \Delta Z_{i-1}|, |Z \Delta S_{i+1}| \geq \ell$. By definition of D , we also have $D[i, Z] = \top$. This completes the induction step. Thus, there is a $Z \in \mathcal{F}_\tau$ such that $D[\tau, Z] = \top$ and if J is a *yes*-instance, then we report that. \blacktriangleleft

4 Application: Committee Election

Bredereck et al. [11] studied the following problem under the name REVOLUTIONARY MULTISTAGE PLURALITY VOTING.

DIVERSE MULTISTAGE PLURALITY VOTING

Input: A set A of agents, a set C of candidates, a sequence $(u_i)_{i=1}^\tau$ of voting profiles $u_i: A \rightarrow C \cup \{\emptyset\}$, and integers $k, x, \ell \in \mathbb{N}$.

Question: Is there a sequence $(C_1, C_2, \dots, C_\tau)$ such that for all $i \in [\tau]$ it holds that $C_i \subseteq C$, $|C_i| \leq k$, and $|u^{-1}(C_i)| \geq x$, and for all $i \in [\tau-1]$ it holds true that $|C_i \Delta C_{i+1}| \geq \ell$?

In this section, we affirmatively answer the question of Bredereck et al. [11] whether DIVERSE MULTISTAGE PLURALITY VOTING parameterized by ℓ or k is in FPT.³

► **Theorem 12.** *An instance J of DIVERSE MULTISTAGE PLURALITY VOTING can be solved in $2^{\mathcal{O}(\ell)} \cdot |J|^{\mathcal{O}(1)}$ time.*

To prove Theorem 12, we use Theorem 1. In the notation of our framework, we deal with the following problem Π : given an instance $I = (A, C, u, k, x)$ consisting of a set A of agents, a set $C =: \mathcal{B}(I)$ of candidates, a voting profile $u: A \rightarrow C$, and two integers k, x , decide whether $\mathcal{R}(I) := \{S \subseteq C \mid k \geq |S| \text{ and } |u^{-1}(S)| \geq x\}$ is non-empty. Hence, to apply Theorem 1, we consider the following problem.

4-COLORED EXACT PLURALITY VOTING

Input: A set A of agents, a set C of candidates, a voting profile $u: A \rightarrow C \cup \{\emptyset\}$, a coloring $c: C \rightarrow [4]$, and integers $n_i, x, k \in \mathbb{N}, i \in [4]$.

Output: A set $C' \subseteq C$ of at most k candidates so that $|u^{-1}(C')| \geq x$ and $|c^{-1}(i) \cap C'| = n_i$ for all $i \in [4]$ or “no” if no such set exists.

This problem is polynomial-time solvable and hence the following observation together with Theorem 1 proves Theorem 12. In the full version we will generalize this application to independent sets in matroids.

► **Observation 13** (\star^4). 4-COLORED EXACT PLURALITY VOTING is polynomial-time solvable.

³ Note that $\ell \leq 2k$ for all non-trivial instances, so it suffices to prove this for ℓ .

⁴ Proofs of results marked with a star are deferred to the full version.

5 Application: Perfect Matching

In this section, we apply our framework from Section 3 to find a sequence of diverse perfect matchings.

DIVERSE MULTISTAGE PERFECT MATCHING

Input: A sequence $(G_i)_{i=1}^\tau$ of graphs and an integer $\ell \in \mathbb{N}_0$.

Question: Is there a sequence $(M_i)_{i=1}^\tau$ of perfect matchings $M_i \subseteq E(G_i)$ such that $|M_i \Delta M_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$?

There are two closely related variants of this problem which were studied extensively. The first variant is the non-diverse variant, where one seeks to bound the symmetric differences (in some way) from above [5, 4, 13, 25, 34]. Steinhau [34] proved that if the size of the symmetric difference of two consecutive perfect matchings shall be at most ℓ , then this problem variant is NP-hard even if ℓ is constant, and W[1]-hard when parameterized by $\ell + \tau$. The second variant is the non-multistage variant, where one is given a single graph and is asked to compute a set of pairwise diverse perfect matchings [22, 23]. Fomin et al. [22] proved that this variant is NP-hard even if one asks only for two diverse matchings. This directly implies NP-hardness for DIVERSE MULTISTAGE PERFECT MATCHING even when $\tau = 2$.

Our goal is to show fixed-parameter tractability of DIVERSE MULTISTAGE PERFECT MATCHING when parameterized by ℓ . This stands in contrast to the NP-hardness for the non-diverse problem variant with constant ℓ .

► **Theorem 14 (★).** *An instance J of DIVERSE MULTISTAGE PERFECT MATCHING can be solved in $2^{O(\ell)} \cdot |J|^{O(1)}$ time with a constant error probability.*

We will prove Theorem 14 by means of Theorem 3 at the end of this section. To this end we need to consider the following problem.

s -COLORED EXACT PERFECT MATCHING

Input: A graph $G = (V, E)$, a coloring $c: E \rightarrow [s]$, and $k_i \in \mathbb{N}, i \in [s]$.

Output: (if exists) A perfect matching M in G such that $|c^i \cap M| = k_i$, for all $i \in [s]$?

For $s = 2$, this problem is known as EXACT MATCHING, and Mulmuley et al. [32] showed that this special case is solvable by a randomized polynomial-time algorithm. We generalize this result by showing that s -COLORED EXACT PERFECT MATCHING can be solved in polynomial time for any constant s by a randomized algorithm with constant error probability. While we only need this for $s = 4$ in order to prove Theorem 14, we believe that the general case may be of independent interest. We remark that it is open whether EXACT MATCHING can be solved in (deterministic) polynomial time.

► **Lemma 15 (★).** *For every $0 < p < 1$ there is an $(n^{O(s)} \cdot \log 1/p)$ -time algorithm which, given an instance of s -COLORED EXACT PERFECT MATCHING, finds a solution with probability at least $1 - p$ if one exists, and concludes that there is no solution otherwise.*

To determine whether a given s -COLORED EXACT PERFECT MATCHING has a solution we use the following algorithm.

► **Algorithm 16.** Let $0 < p < 1$ and let $I = (G, c, k_1, \dots, k_s)$ be an instance of s -COLORED EXACT PERFECT MATCHING where $G = (V, E)$ has n vertices.

Step 1. Set $\gamma := \lceil n/(2p) \rceil$ and draw $w_{ij} \in [\gamma]$ for all $\{i, j\} \in E$ uniformly at random.

Step 2. Construct an $n \times n$ matrix A' with entries $a_{ij} \in \mathbb{Z}[y_1, \dots, y_s]$, $1 \leq i \leq j \leq n$, where

$$a_{ij} := \begin{cases} 0 & \text{if } \{i, j\} \notin E, \\ w_{ij}y_q & \text{if } \{i, j\} \in c^q \cap E, q \in [s]. \end{cases}$$

Afterwards we compute the skew-symmetric matrix $A := A' - (A')^T$.

Step 3. Compute the polynomial $P := \sqrt{\det(A)} \in \mathbb{Z}[y_1, \dots, z_s]$.

Step 4. If P contains a monomial $b^*y_1^{k_1}y_2^{k_2}\dots y_s^{k_s}$ such that $b^* \neq 0$ then, output *yes*. Otherwise, output *no*. \diamond

Before studying the running time of Algorithm 16, we first focus on its correctness.

► **Lemma 17.** *Let I and p be the input of Algorithm 16. If Algorithm 16 returns *yes*, then there is a solution for I . Conversely, if I is a *yes*-instance, then Algorithm 16 returns *yes* with probability at least $1 - p$.*

Proof. Let \mathcal{P} be the set of all partitions of V into unordered pairs. For $\sigma \in \mathcal{P}$ with $\sigma = \{\{i_1, j_1\}, \{i_2, j_2\}, \dots, \{i_{n/2}, j_{n/2}\}\}$ with $i_k < j_k$ for $k \in [n/2]$ and $i_1 < i_2 < \dots < i_{n/2}$, let

$$\pi_\sigma := \begin{bmatrix} 1 & 2 & 3 & 4 & \dots & n-1 & n \\ i_1 & j_1 & i_2 & j_2 & \dots & i_{n/2} & j_{n/2} \end{bmatrix}$$

be the corresponding permutation. Let $\text{val}(\sigma) := \text{sgn}(\pi_\sigma) \prod_{\{i,j\} \in \sigma} a_{ij}$, where $\text{sgn}(\pi_\sigma) \in \{+1, -1\}$ is the signum of π_σ . The Pfaffian of A (computed by Algorithm 16) is defined as $\text{pf}(A) := \sum_{\sigma \in \mathcal{P}} \text{val}(\sigma)$ [27]. Note that A is skew-symmetric, hence, $\text{pf}(A) = \sqrt{\det(A)} = P$ [31, 27]. As $\text{val}(\sigma) = 0$ whenever σ contains a non-edge, we have $P = \sum_{M \in \mathcal{PM}} \text{val}(M)$, where \mathcal{PM} is the set of perfect matchings in G . Let M be a perfect matching and let $z_q = |c^q \cap M|$, $q \in [s]$. Then $\text{val}(M) = \text{sgn}(\pi_M) \prod_{q \in [s]} \prod_{\{i,j\} \in M \cap c^q} w_{ij}y_q = b \cdot y_1^{z_1}y_2^{z_2}\dots y_s^{z_s}$, where $b \in \mathbb{Z}$. Let $\mathcal{PM}^* \subseteq \mathcal{PM}$ be the family of perfect matchings M^* which have exactly k_i edges of color i , for all $i \in [s]$. Then the coefficient b^* of the monomial $b^*y_1^{k_1}y_2^{k_2}\dots y_s^{k_s}$ of P is $b^* = \sum_{M^* \in \mathcal{PM}^*} \text{sgn}(\pi_{M^*}) \prod_{\{i,j\} \in M^*} w_{ij}$. Hence, if Algorithm 16 returns *yes* (i.e., $b^* \neq 0$), then $\mathcal{PM}^* \neq \emptyset$.

Now conversely assume I to be a *yes*-instance, i.e., $\mathcal{PM}^* \neq \emptyset$. We analyze the probability of the event $b^* = 0$ occurring. Note that b^* can be seen as a polynomial of degree at most $n/2$ over the indeterminates $\{w_{ij} \mid \{i, j\} \in E\}$. As we have drawn the w_{ij} independently and uniformly at random from $[\gamma]$ with $\gamma \geq n/(2p)$, by the DeMillo-Lipton-Schwartz-Zippel lemma the probability that $b^* = 0$ is at most $n/(2\gamma) \leq p$. \blacktriangleleft

Now we show that Algorithm 16 can be executed efficiently.

► **Lemma 18 (★).** *Algorithm 16 runs in $n^{O(s)} \log(1/p)$ time.*

We are now ready to put all parts together and prove Lemma 15. In a nutshell, we use Algorithm 16 to check whether there is a solution. If this is the case, then we try to delete as many edges as possible from the instance until the whole edge set is a solution. Putting Lemma 15 and Theorem 3 together, we can prove the main theorem of this section, see the full version for details.

6 Application: s - t Path

In this section, we apply our framework to the task of finding a sequence of diverse s - t paths. This has obvious applications e.g. in convoy routing [21].

DIVERSE MULTISTAGE s - t PATH

Input: A sequence of graphs $(G_i)_{i=1}^\tau$, two distinct vertices $s, t \in \bigcap_{i=1}^\tau V(G_i)$, and $\ell \in \mathbb{N}_0$.

Question: Is there a sequence $(P_1, P_2, \dots, P_\tau)$ such that P_i is an s - t path in G_i for all $i \in [\tau]$, and $|S_i \Delta S_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$?

Our goal is to show that DIVERSE MULTISTAGE s - t PATH parameterized by ℓ is in FPT.

► **Theorem 19.** DIVERSE MULTISTAGE s - t PATH parameterized by ℓ is in FPT.

We will prove Theorem 19 by means of Theorem 1 at the end of this section. To this end, we need to consider the following problem.

4-COLORED EXACT s - t PATH

Input: A graph G , distinct vertices $s, t \in V(G)$, coloring $c: V(G) \rightarrow [4]$, and $n_i \in \mathbb{N}_0, i \in [4]$.

Output: (if exists) An s - t path P such that $|c^{-1}(i) \cap V(P)| = n_i$ for all $i \in [4]$.

Unfortunately, 4-COLORED EXACT s - t PATH is unlikely to be polynomial-time solvable, as it is NP-hard even if only a single color is used, by a trivial reduction from HAMILTONIAN PATH. However, as we will see in the proof of Theorem 19, by a result of Mousset et al. [30] we can actually reduce 4-COLORED EXACT s - t PATH to the case that all graphs have small treewidth. In this setting, we then employ dynamic programming.

► **Lemma 20 (★).** 4-COLORED EXACT s - t PATH is solvable $k^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time, where k is the treewidth of the input graph G .

While some techniques [15, 24, 9] seem applicable to improve the running time of Lemma 20 slightly, for our needs a straight-forward dynamic program on a nice tree decomposition suffices. We are now ready to prove Theorem 19.

Proof of Theorem 19. Let the instance J of DIVERSE MULTISTAGE s - t PATH be given in the form of graphs G_1, \dots, G_τ , two vertices $s, t \in \bigcap_{i=1}^\tau G_i$ and $\ell \in \mathbb{N}$. We may assume that every vertex v of every graph G_i is contained in at least one s - t path in G_i , since otherwise we may delete v . This is equivalent to the assumption that the graph G'_i obtained from adding the edge $\{s, t\}$ to G_i is biconnected.

By a result of Mousset et al. [30], there is a universal constant $\gamma > 0$, such that each G_i with treewidth $\text{tw}(G_i) \geq \gamma\ell$ contains two vertex-disjoint cycles of size at least 4ℓ .

If two such cycles C, C' exist in G_i , then let P_1 be an s - t path containing at least one edge of C . To see that such a path exists, construct a biconnected graph by simply attaching a new degree-two vertex s' to both s and t , create another new vertex t' by subdividing some edge of C , and take two disjoint paths between s' and t' .

Without loss of generality, P_1 enters C and C' at most once each. Construct another s - t path P_2 from P_1 by setting $E(P_2) := E(P_1) \Delta E(C)$. If P_1 contains any edge of C' , then define P_3 by $E(P_3) := E(P_1) \Delta E(C')$. Otherwise, let P_3 be any s - t path containing at least half of the edges of C' (this can be achieved analogously to the construction of P_1 resp. P_2). Observe that P_1, P_2 , and P_3 have pairwise symmetric differences at least 2ℓ . Thus, $\{P_1, P_2, P_3\}$ is an ℓ -diverse representative of all s - t paths in G_i by Lemma 5.

We can then solve the subinstances given by $(G_j)_{j < i}$ and $(G_j)_{j > i}$ separately and pick a suitable path from $\{P_1, P_2, P_3\}$ afterwards.

All subinstances in which every graph G_i has $\text{tw}(G_i) < \gamma\ell$ can be solved by Theorem 1 in combination with Lemma 20 in $2^{\mathcal{O}(\ell)} f(\gamma\ell) |J|^{\mathcal{O}(1)}$ time, where f is given by Lemma 20. ◀

7 Hardness of Vertex Cover

We finally present a problem where our framework from Section 3 is not applicable, unless $\text{FPT} = \text{W}[1]$. The non-diverse variant of the following problem was studied by Fluschnik et al. [20]. Among others, they showed $\text{W}[1]$ -hardness when parametrized by the vertex cover size k or by the maximum number of edges over all instances in the input.

DIVERSE MULTISTAGE VERTEX COVER

Input: A sequence of graphs $(G_i)_{i=1}^\tau$ and $k, \ell \in \mathbb{N}$.

Question: Is there a sequence $(S_1, S_2, \dots, S_\tau)$ such that for all $i \in [\tau]$ the set $S_i \subseteq V(G_i)$ is a vertex cover of size at most k in G_i and $|S_i \Delta S_{i+1}| \geq \ell$ for all $i \in [\tau - 1]$?

The framework from Section 3 is presumably not applicable to DIVERSE MULTISTAGE VERTEX COVER because of the following result.

► **Theorem 21.** DIVERSE MULTISTAGE VERTEX COVER *parameterized by ℓ is $\text{W}[1]$ -hard, even if $\tau = 2$.*

Proof. We reduce from INDEPENDENT SET: Given a graph $G = (V, E)$, and $k \in \mathbb{N}$, is there a vertex set $S \subseteq V$, $|S| \geq k$, such that the vertices in S are pairwise nonadjacent? INDEPENDENT SET is $\text{W}[1]$ -hard with respect to k [17].

Let $I := (G = (V, E), k)$ be an instance of INDEPENDENT SET and let $|V| = n$. Without loss of generality, we assume that $k > 1$. We construct an instance $J := ((G_1, G_2), k', \ell)$ of DIVERSE MULTISTAGE VERTEX COVER as follows. The first graph G_1 is a complete graph on the vertex set $V \cup \{v\}$. The second graph G_2 consists of the vertex set $V \cup \{v\}$ and the edge set $E \cup \{\{u, v\} \mid u \in V\}$, that is, G_2 is a copy of G to which we add a vertex v which is adjacent to every other vertex. Lastly, we set $k' := n$ and $\ell := k + 1$. Clearly, J can be constructed in polynomial time. We now show that I is a *yes*-instance if and only if J is a *yes*-instance.

(\Rightarrow): Let S be an independent set of size at least k in G . Let $S_1 := V$ and $S_2 := \{v\} \cup V \setminus S$. Note that $|S_1 \Delta S_2| \geq k + 1$ and $|S_2| \leq |S_1| = n$, and S_i is a vertex cover in G_i for $i \in [2]$. Thus (S_1, S_2) is a valid solution for our instance of DIVERSE MULTISTAGE VERTEX COVER.

(\Leftarrow): Let (S_1, S_2) be a solution for our instance of DIVERSE MULTISTAGE VERTEX COVER. As G_1 is a complete graph, we have $|S_1| \geq n$. Without loss of generality, we assume that $S_1 = V$. Then $S_1 \Delta S_2 = \{v\} \cup V \setminus S_2$. Note that $v \in S_2$, otherwise S_2 must be equal to V in order to be a vertex cover, and $|S_1 \Delta V| < \ell$. As S_2 is a vertex cover of G_2 , the set $S := (V \cup \{v\}) \setminus S_2$ is an independent set of G_2 . Note that $v \notin S$, hence S is also an independent set of G . Finally, as $S = (S_1 \Delta S_2) \setminus \{v\}$, we have $|S| \geq \ell - 1 = k$, and we are done. ◀

8 Conclusion

We introduced a versatile framework to show fixed-parameter tractability for a variety of diverse multistage problems when parameterized by the diversity ℓ . The only requirement for applying our framework is that a four-colored variant of the base problem can be solved efficiently. We presented four applications of our framework, one of which resolving an open question by Brederick et al. [11]. Two other applications revealed problems which may be of independent interest from a technical and motivational point of view, see Sections 5 and 6.

We believe that our framework can be applied to a broad spectrum of multistage problems. In particular, a broad systematic study of the multistage setting in elections was proposed by Boehmer and Niedermeier et al. [10]. Herein, diversity is a natural goal. From a motivational

point of view, an interesting direction for future research is to combine the diverse multistage setting with time windows, known from other temporal domains [35, 28, 12, 1, 29]. Here, a solution to the i -th instance should be sufficiently different from the δ previous solutions in the sequence; our work covers the case $\delta = 1$. In some multistage scenarios a “global view” [26] on the symmetric differences is desired. In context of this paper this means that two consecutive solutions can have a small symmetric difference as long as the sum of all consecutive symmetric differences is at least ℓ . We believe that our framework (Section 3) can be extended to this setting. To see this, we have to realize that for an ℓ -diverse representative \mathcal{F} of a family of solutions the following holds: For all sets A and B and integers $\ell_a, \ell_b \leq \ell$, if there is an $S \in \mathcal{F}$ such that $|A \Delta S| \geq \ell_a$ and $|B \Delta S| \geq \ell_b$, then there is an $\hat{S} \in \mathcal{F}$ such that $|A \Delta \hat{S}| \geq \ell_a$ and $|B \Delta \hat{S}| \geq \ell_b$. We leave the details for further research. Finally, the presented time and space constraints to compute ℓ -diverse representatives seem to be suboptimal. Hence, improving the time or space constraints could be a fruitful research direction.

References

- 1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 3 Emmanuel Arrighi, Henning Fernau, Daniel Lokshantov, Mateus de Oliveira Oliveira, and Petra Wolf. Diversity in kemeny rank aggregation: A parameterized approach, 2021. arXiv:2105.09413.
- 4 Evripidis Bampis, Bruno Escoffier, and Alexander V. Kononov. LP-based algorithms for multistage minimization problems, 2019. arXiv:1909.10354.
- 5 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. Multistage matchings. In *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 101, pages 7:1–7:13, 2018. doi:10.4230/LIPIcs.SWAT.2018.7.
- 6 Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. Online multistage subset maximization problems. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, volume 144, pages 11:1–11:14, 2019. doi:10.4230/LIPIcs.ESA.2019.11.
- 7 Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage knapsack. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 138, pages 22:1–22:14, 2019. doi:10.4230/LIPIcs.MFCS.2019.22.
- 8 Julien Baste, Michael R. Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geervarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1119–1125, 2020. doi:10.24963/ijcai.2020/156.
- 9 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
- 10 Niclas Boehmer and Rolf Niedermeier. Broadening the research agenda for computational social choice: Multiple preference profiles and multiple solutions. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1–5, 2021.
- 11 Robert Brederick, Till Fluschnik, and Andrzej Kaczmarczyk. Multistage committee election, 2020. arXiv:2005.02300.
- 12 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. In *Proceedings of the 31st International*

- Symposium on Algorithms and Computation (ISAAC)*, volume 181, pages 30:1–30:18, 2020. doi:10.4230/LIPIcs.ISAAC.2020.30.
- 13 Markus Chimani, Niklas Troost, and Tilo Wiedera. Approximating multistage matching problems, 2020. arXiv:2002.06887.
 - 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
 - 15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.
 - 16 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.
 - 17 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
 - 18 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 459–470, 2014. doi:10.1007/978-3-662-43951-7_39.
 - 19 Till Fluschnik. A multistage view on 2-satisfiability. In *Proceedings of the 11th International Conference on Algorithms and Complexity (CIAC)*, 2021. To appear. arXiv:2011.02325.
 - 20 Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. Multistage vertex cover. In *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 148, pages 14:1–14:14, 2019. doi:10.4230/LIPIcs.IPEC.2019.14.
 - 21 Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. Multistage s-t path: Confronting similarity with dissimilarity in temporal graphs. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181, pages 43:1–43:16, 2020. doi:10.4230/LIPIcs.ISAAC.2020.43.
 - 22 Fedor V. Fomin, Petr A. Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181, pages 26:1–26:12, 2020. doi:10.4230/LIPIcs.ISAAC.2020.26.
 - 23 Fedor V. Fomin, Petr A. Golovach, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Diverse collections in matroids and graphs. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 187, pages 31:1–31:14, 2021. doi:10.4230/LIPIcs.STACS.2021.31.
 - 24 Fedor V. Fomin, Daniel Lokshantov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA)*, volume 8737, pages 443–454, 2014. doi:10.1007/978-3-662-44777-2_37.
 - 25 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8572, pages 563–575, 2014. doi:10.1007/978-3-662-43948-7_47.
 - 26 Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theoretical Computer Science*, 868:46–64, 2021. doi:10.1016/j.tcs.2021.04.002.
 - 27 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009.
 - 28 George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154, pages 27:1–27:14, 2020. doi:10.4230/LIPIcs.STACS.2020.27.

- 29 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. In *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, pages 7667–7674, 2019. doi:10.1609/aaai.v33i01.33017667.
- 30 Frank Mousset, Andreas Noever, Nemanja Škoric, and Felix Weissenberger. A tight Erdős-Pósa function for long cycles. *Journal of Combinatorial Theory Series B*, 125:21–32, 2017. doi:10.1016/j.jctb.2017.01.004.
- 31 Thomas Muir. *A Treatise on the Theory of Determinants*. MacMillan and Co., London, 1882.
- 32 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.
- 33 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191, 1995. doi:10.1109/SFCS.1995.492475.
- 34 Lino Steinhau. Parameterized algorithmics of multistage matching. Bachelor thesis, Technische Universität Berlin, 2020.
- 35 Philipp Zschoche. A faster parameterized algorithm for temporal matching, 2021. arXiv: 2010.10408.

Fast and Space-Efficient Construction of AVL Grammars from the LZ77 Parsing

Dominik Kempa 

Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Ben Langmead 

Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Abstract

Grammar compression is, next to Lempel–Ziv (LZ77) and run-length Burrows–Wheeler transform (RLBWT), one of the most flexible approaches to representing and processing highly compressible strings. The main idea is to represent a text as a context-free grammar whose language is precisely the input string. This is called a straight-line grammar (SLG). An AVL grammar, proposed by Rytter [Theor. Comput. Sci., 2003] is a type of SLG that additionally satisfies the AVL property: the heights of parse trees for children of every nonterminal differ by at most one. In contrast to other SLG constructions, AVL grammars can be constructed from the LZ77 parsing in compressed time: $\mathcal{O}(z \log n)$ where z is the size of the LZ77 parsing and n is the length of the input text. Despite these advantages, AVL grammars are thought to be too large to be practical.

We present a new technique for rapidly constructing a small AVL grammar from an LZ77 or LZ77-like parse. Our algorithm produces grammars that are always at least five times smaller than those produced by the original algorithm, and usually not more than double the size of grammars produced by the practical Re-Pair compressor [Larsson and Moffat, Proc. IEEE, 2000]. Our algorithm also achieves low peak RAM usage. By combining this algorithm with recent advances in approximating the LZ77 parsing, we show that our method has the potential to construct a run-length BWT in about one third of the time and peak RAM required by other approaches. Overall, we show that AVL grammars are surprisingly practical, opening the door to much faster construction of key compressed data structures.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Pattern matching

Keywords and phrases grammar compression, straight-line program, SLP, AVL grammar, Lempel–Ziv compression, LZ77, dictionary compression

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.56

Related Version *Full Version*: <https://arxiv.org/abs/2105.11052>

Supplementary Material *Software*: <https://github.com/dominikkempa/lz77-to-slp>

Funding DK and BL were supported by NIH HG011392 and NSF DBI-2029552 grants.

Acknowledgements We thank Simon J. Puglisi for providing us with the kernel data set at short notice.

1 Introduction

The increase in the amount of highly compressible data that requires efficient processing in the recent years, particularly in the area of computational genomics [3, 4], has caused a spike of interest in dictionary compression. Its main idea is to reduce the size of the representation of data by finding repetitions in the input and encoding them as references to other occurrences. Among the most popular methods are the Lempel–Ziv (LZ77) compression [32], run-length Burrows–Wheeler transform (RLBWT) [5, 16], and grammar compression [6]. Although in theory, LZ77 and RLBWT are separated by at most a factor of $\mathcal{O}(\log^2 n)$ (where n is the length



© Dominik Kempa and Ben Langmead;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 56; pp. 56:1–56:14

Leibniz International Proceedings in Informatics



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

of the input text) [15, 20], the gap in practice is usually noticeable (as also confirmed by our experiments). RLBWT is the largest of the three representations in practice, but is also the most versatile, supporting powerful suffix array and suffix tree queries [16]. LZ77, on the other hand, is the smallest, but its functionality includes only the easier longest common extension (LCE), random-access, and pattern matching queries [1, 7, 12, 13, 21]. Grammar compression occupies the middle ground between the two, supporting queries similar to LZ77 [26]. Navarro gives a comprehensive overview of these and related representations [26, 27].

A major practical concern with these representations – RLBWT in particular – is how to construct them efficiently. Past efforts have focused on engineering efficient *general* algorithms for constructing the BWT and LZ77 [10, 18, 2, 17], but these are not applicable to the terabyte-scale datasets routinely found, e.g., in modern genomics [4]. Specialized algorithms for highly repetitive datasets have only been investigated recently. Boucher et al. [4] proposed a method for the efficient construction of RLBWT using the concept of prefix-free parsing. The same problem was approached by Policriti and Prezza, and Ohno et al. [29, 28], using a different approach based on the dynamic representation of RLBWT. These methods represent the state of the art in the practical construction of RLBWT.

A different approach to the construction of RLBWT was recently proposed in [20]. The idea is to first compute the (exact or approximate) LZ77 parsing for the text, and then convert this representation into an RLBWT. Crucially, the LZ77 \rightarrow RLBWT conversion takes only $\mathcal{O}(z \text{ polylog } n)$ time (where z is the size of the LZ77 parsing), i.e., it runs not only in the compressed space but also in *compressed time*.¹ The computational bottleneck is therefore shifted to the easier problem of computing or approximating the LZ77, which is the only step taking $\Omega(n)$ time. Internally, this new pipeline consists of three steps: text \rightarrow (approximate) LZ77 \rightarrow grammar \rightarrow RLBWT, unsurprisingly aligning with the gradual increase in the size and complexity of these representations. Kosolobov et al. [24] recently proposed a fast and space-efficient algorithm to approximate LZ77, called Re-LZ. The second and third steps in the pipeline, from the LZ77 parse to the RLBWT, have not been implemented. The only known algorithm to convert LZ77 into a grammar in compressed time was proposed by Rytter [30], and is based on the special type of grammars called *AVL grammars*, whose distinguishing feature is that all subtrees in the parse tree satisfy the AVL property: the tree-heights for children of every nonterminal do not differ by more than one. The algorithm is rather complex, and until now has been considered impractical.

Our Contribution. Our main contribution is a series of practical improvements to the basic variant of Rytter’s algorithm, and a fast and space-efficient implementation of this improved algorithm. Compared to the basic variant, ours produces a grammar that is always five times smaller, and crucially, the same holds for all intermediate grammars computed during the algorithm’s execution, yielding very low peak RAM usage. The resulting grammar is also no more than twice of the smallest existing grammar compressors such as Re-Pair [25]. We further demonstrate that combining our new improved algorithm with Re-LZ opens up a new path to the construction of RLBWT. Our preliminary experiments indicate that at least a three-fold speedup and the same level of reduction in the RAM usage is possible.

The key algorithmic idea in our variant is to delay the merging of intermediate AVL grammars as much as possible to avoid creating nonterminals that are then unused in the final grammar. We dub this variant *lazy AVL grammars*. We additionally incorporate Karp–Rabin fingerprints [19] to re-write parts of the grammar on-the-fly and further reduce

¹ $\text{polylog } n = \log^c n$ for some constant $c > 0$.

the grammar size. We describe two distinct versions of this technique: greedy and optimal, and demonstrate that both lead to reductions in the grammar size. As a side-result of independent interest, we describe a fast and space-efficient data structure for the dynamic predecessor problem, in which the inserted key is always larger than all other elements currently in the set.

2 Preliminaries

Strings. For any string S , we write $S[i..j]$, where $1 \leq i, j \leq |S|$, to denote a substring of S . If $i > j$, we assume $S[i..j]$ to be the empty string ε . By $[i..j]$ we denote $[i..j-1]$. Throughout the paper, we consider a string (text) $T[1..n]$ of $n \geq 1$ symbols from an integer alphabet $\Sigma = [0..\sigma)$. By $\text{LCE}(i, i')$ we denote the length of the longest common prefix of suffixes $T[i..n]$ and $T[i'..n]$.

Karp–Rabin Fingerprints. Let q be a prime number and let $r \in [0..q)$ be chosen uniformly at random. The *Karp–Rabin fingerprint* of a string S is defined as

$$\Phi(S) = \sum_{i=1}^{|S|} S[i] \cdot r^{|S|-i} \bmod q.$$

Clearly, if $T[i..i+\ell] = T[j..j+\ell]$ then $\Phi(T[i..i+\ell]) = \Phi(T[j..j+\ell])$. On the other hand, if $T[i..i+\ell] \neq T[j..j+\ell]$ then $\Phi(T[i..i+\ell]) \neq \Phi(T[j..j+\ell])$ with probability at least $1 - \ell/q$ [9]. In our algorithm we are comparing only substrings of T of equal length. Thus, the number of different possible substring comparisons is less than n^3 , and hence for any positive constant c , we can set q to be a prime larger than n^{c+4} (but still small enough to fit in $\mathcal{O}(1)$ words) to make the fingerprint function perfect with probability at least $1 - n^{-c}$.

LZ77 Compression. An *LZ77-like factorization* of T is a factorization $T = F_1 \cdots F_f$ into non-empty *phrases* such that every phrase F_j with $|F_j| > 1$ has an earlier occurrence in T , i.e., letting $i = 1 + |F_1 \cdots F_{j-1}|$ and $\ell = |F_j|$, there exists $p \in [1..i)$ satisfying $\text{LCE}(p, i) \geq \ell$. The phrase $F_j = T[i..i+\ell]$ is encoded as a pair (p, ℓ) . If there are multiple choices for p , we choose one arbitrarily. The occurrence $T[p..p+\ell]$ is called the *source* of F_j . If $\ell = 1$, the phrase $F_j = T[i]$ is encoded as a pair $(T[i], 0)$. The LZ77-like parsing, in which we additionally require the phrase to not overlap its source, i.e., $p + \ell \leq i$, is called *non-self-referential*.

The LZ77 factorization [32] (or the LZ77 parsing) of a string T is an LZ77-like factorization constructed by greedily parsing T from left to right into longest possible phrases. More precisely, the j th phrase F_j is the longest substring starting at position $i = 1 + |F_1 \cdots F_{j-1}|$ that has an earlier occurrence in T . If there is no such substring, then $F_j = T[i]$. We denote the number of phrases in the LZ77 parsing by z . For example, the text `bbabaababababaababa` has the LZ77 parsing `b · b · a · ba · aba · bababa · ababa` with $z = 7$ phrases, and is encoded as a sequence $(b, 0), (1, 1), (a, 0), (2, 2), (3, 3), (7, 6), (10, 5)$.

Grammar Compression. A context-free grammar is a tuple $G = (N, \Sigma, R, S)$, where N is a finite set of *nonterminals*, Σ is a finite set of *terminals*, and $R \subseteq N \times (N \cup \Sigma)^*$ is a set of *rules*. We assume $N \cap \Sigma = \emptyset$ and $S \in N$. The nonterminal S is called the *starting symbol*. If $(A, \gamma) \in R$ then we write $A \rightarrow \gamma$. The *language* of G is set $L(G) \subseteq \Sigma^*$ obtained by starting with S and repeatedly replacing nonterminals with their expansions, according to R .

A grammar $G = (N, \Sigma, R, S)$ is called a *straight-line grammar* (SLG) if for any $A \in N$ there is exactly one production with A of the left side, and all nonterminals can be ordered $A_1, \dots, A_{|N|}$ such that $S = A_1$ and if $A_i \rightarrow \gamma$ then $\gamma \in (\Sigma \cup \{A_{i+1}, \dots, A_{|N|}\})^*$, i.e., the

graph of grammar rules is acyclic. The unique γ such that $A \rightarrow \gamma$ is called the *definition* of A and is denoted $\text{rhs}(A)$. In any SLG, for any $u \in (N \cup \Sigma)^*$ there exists exactly one $w \in \Sigma^*$ that can be obtained from u . We call such w the *expansion* of u , and denote it by $\text{exp}(u)$. We define the *parse tree* of $A \in N \cup \Sigma$ as a rooted ordered tree $\mathcal{T}(A)$, where each node v is associated to a symbol $\text{sym}(v) \in N \cup \Sigma$. The root of $\mathcal{T}(A)$ is a node v such that $\text{sym}(v) = A$. If $A \in \Sigma$ then v has no children. If $A \in N$ and $\text{rhs}(A) = B_1 \cdots B_k$, then v has k children and the subtree rooted at the i th child is a copy of $\mathcal{T}(B_i)$. The parse tree $\mathcal{T}(G)$ is defined as $\mathcal{T}(S)$. The *height* of any $A \in N$ is defined as the height of $\mathcal{T}(A)$, and denoted $\text{height}(A)$.

The idea of *grammar compression* is, given a text T , to compute a small SLG G such that $L(G) = \{T\}$. The *size* of the grammar is measured by the total length of all definitions, and denoted $|G| := \sum_{A \in N} |\text{rhs}(A)|$. Clearly, it is easy to encode any G in $\mathcal{O}(|G|)$ space: pick an ordering of nonterminals and write down the definitions of all nonterminals, replacing nonterminal symbols with their numbers in the order.

3 AVL Grammars and the Basic Algorithm

An SLG $G = (N, \Sigma, R, S)$ is said to be in *Chomsky normal form*, if for every $A \in N$, it holds $\text{rhs}(A) \in \Sigma$ or $\text{rhs}(A) = XY$, where $X, Y \in N$. An SLG in Chomsky normal form is called a *straight-line program* (SLP). Rytter [30] defines an *AVL grammar* as an SLP $G = (N, \Sigma, R, S)$ that additionally satisfies the *AVL property*: for every $A \in N$ such that $\text{rhs}(A) = XY$, it holds $|\text{height}(X)| - |\text{height}(Y)| \leq 1$. This condition guarantees that for every $A \in N$ (in particular for $S \in N$), it holds $\text{height}(A) = \mathcal{O}(\log |\text{exp}(A)|)$ [30, Lemma 1].

The main result presented in [30] is an algorithm that given a non-self-referential LZ77-like parsing of a length- n text T consisting of f phrases, computes in $\mathcal{O}(f \log n)$ time an AVL grammar G generating T and satisfying $|G| = \mathcal{O}(f \log n)$. Rytter's construction was extended to allow self-references in [20, Theorem 6.1]. Our implementation of the basic as well as improved Rytter's algorithm works for the self-referential variant, but for simplicity here and in Section 4 we describe the algorithm only for the non-self-referential variant.

The algorithm in [30] works in f steps. It maintains the dynamically changing AVL grammar G such that after the k th step is complete, there exists a nonterminal P_k in G such that $\text{exp}(P_k) = F_1 \cdots F_k$, where $T = F_1 \cdots F_f$ is the input LZ77-like factorization of the input. This implies that at end there exist a nonterminal expanding to T . The algorithm does not delete any nonterminals between steps. At the end, it may perform an optional pruning of the grammar to remove the nonterminals not present in the parse tree $\mathcal{T}(P_f)$. This reduces the grammar size but not the peak memory usage of the algorithm.

The algorithm uses the following three procedures, each of which adds a nonterminal A with a desired expansion $\text{exp}(A)$ to the grammar G , along with a bounded number of extra nonterminals:

1. **AddSymbol(c)**: Given $c \in \Sigma$, add a nonterminal A with $\text{rhs}(A) = c$ to the grammar G .
2. **AddMerged(X, Y)**: Given the identifiers of nonterminals X, Y existing in G , add a nonterminal A to G that satisfies $\text{exp}(A) = \text{exp}(X)\text{exp}(Y)$. The difficulty of this operation is ensuring that the updated G remains an AVL grammar. Simply setting $\text{rhs}(A) = XY$ would violate the AVL condition in most cases. Instead, the algorithm performs the procedure similar to the concatenation of AVL trees [22, p. 474], taking $\mathcal{O}(1 + |\text{height}(X) - \text{height}(Y)|)$ time, and introducing $\mathcal{O}(|\text{height}(X) - \text{height}(Y)|)$ extra nonterminals.
3. **AddSubstring(A, i, j)**: Given the identifier of a nonterminal A existing in G , and two positions satisfying $1 \leq i \leq j \leq |\text{exp}(A)|$, add a nonterminal B to G that satisfies $\text{exp}(B) = \text{exp}(A)[i..j]$. To explain how this is achieved, we define the auxiliary proce-

cedure $\text{Decompose}(A, i, j)$ that given the same parameters as above, returns the sequence B_1, \dots, B_q of nonterminals satisfying $\exp(A)[i..j] = \exp(B_1) \cdots \exp(B_q)$. The nonterminals B_i are found by performing two root-to-leaf traversals in the parse tree $\mathcal{T}(A)$. This takes $\mathcal{O}(1 + \text{height}(A)) = \mathcal{O}(1 + \log |\exp(A)|)$ time and ensures $q = \mathcal{O}(\log |\exp(A)|)$. It is easy to see that given B_1, \dots, B_q , we can now obtain B in $\mathcal{O}(1 + \log^2 |\exp(A)|)$ time using AddMerged . In [30], it was however shown that if we always choose the shortest nonterminal to merge with its neighbor, the total runtime is $\mathcal{O}(1 + \log |\exp(A)|)$ and only $\mathcal{O}(\log |\exp(A)|)$ extra nonterminals are introduced.

Using the above three procedures, the algorithm in [30] works as follows. Suppose we have already processed the leftmost $k - 1$ phrases. The step begins by creating a nonterminal A_k satisfying $\exp(A_k) = F_k$. If $|F_k| = 1$, it uses the procedure $\text{AddSymbol}(F_k)$. Otherwise, A_k is obtained as the output of $\text{AddSubstring}(P_{k-1}, p, p + \ell - 1)$, where $\ell = |F_k|$ and $T[p..p + \ell]$ is the source of phrase F_k . Finally, P_k is obtained as the output of $\text{AddMerged}(P_{k-1}, A_k)$. A single iteration thus takes $\mathcal{O}(1 + \log |F_1 \cdots F_{k-1}|) = \mathcal{O}(\log n)$ time and adds $\mathcal{O}(\log n)$ extra nonterminals, for total of $\mathcal{O}(f \log n)$ nonterminals over all steps.

4 Modified Algorithm

Lazy Merging. We start by observing that the main reason responsible for the large final grammar produced by the algorithm in Section 3 is the requirement that at the end of each step $k \in [1..f]$, there exist a nonterminal P_k satisfying $\exp(P_k) = F_1 \cdots F_k$. We relax this requirement, and instead require only that at the end of step k , there exists a sequence of nonterminals R_1, \dots, R_m such that $\exp(R_1) \cdots \exp(R_m) = F_1 \cdots F_k$. The algorithm explicitly maintains these nonterminals as a sequence of pairs $(\ell_1, R_1), \dots, (\ell_m, R_m)$, where $\ell_j = \sum_{i=1}^j |\exp(R_i)|$. The modified algorithm uses the following new procedures:

1. **MergeEnclosed**(i, j): Given two positions satisfying $1 \leq i \leq j \leq |F_1 \cdots F_{k-1}|$, add to G a nonterminal R satisfying $\exp(R) = \exp(R_x) \cdots \exp(R_y)$, where $x = \min\{t \in [1..m] : \ell_{t-1} \geq i - 1\}$ and $y = \max\{t \in [1..m] : \ell_t \leq j\}$. The positions x and y are found using a binary search. The pairs of the sequence $(\ell_1, R_1), \dots, (\ell_m, R_m)$ at positions between x and y are then removed and replaced with a pair (ℓ_y, R) . In other words, this procedure merges all the nonterminals from the current root sequence whose expansion is entirely inside the given interval $[i..j]$. Merging of R_x, \dots, R_y is performed pairwise, using the AddMerged procedure. Note, however, that there is no dependence between heights of the adjacent nonterminals (in particular, they do not form a bitonic sequence, like in the algorithm in Section 3), and moreover, their number $\hat{m} := y - x + 1$ is not bounded. To minimize the number of newly introduced nonterminals, we thus employ a greedy heuristic, that always chooses the nonterminal with the smallest height among the remaining elements, and merges it with a shorter neighbor. We use a list to keep the pointers between neighbors and a binary heap to maintain heights. The merging thus runs in $\mathcal{O}(\hat{m} \log n)$ time.
2. **DecomposeWithRoots**(i, j): Given two positions satisfying $1 \leq i \leq j \leq |F_1 \cdots F_{k-1}|$, this procedure returns a sequence of nonterminals A_1, \dots, A_q satisfying $(F_1 \cdots F_{k-1})[i..j] = \exp(A_1) \cdots \exp(A_q)$. First, it computes positions x and y , as defined in the description of MergeEnclosed above. It then returns the result of $\text{Decompose}(R_{x-1}, i - \ell_{x-2}, \ell_{x-1} - \ell_{x-2})$, followed by R_x, \dots, R_y , followed by the result of $\text{Decompose}(R_{y+1}, 1, j - \ell_y)$ (appropriately handling the boundary cases, which for clarity we ignore here). In other words, this procedure finds the sequence of nonterminals that uses as many roots from the sequence R_1, \dots, R_m , as possible, and runs the standard Decompose for the boundary roots. Letting $\hat{m} = y - x + 1$, it runs in $\mathcal{O}(\hat{m} + \log n)$ time.

Using the above additional procedures, our algorithm works as follows. Suppose that we have already processed the first $k - 1$ phrases. The step begins by computing the sequence of nonterminals A_1, \dots, A_q satisfying $\exp(A_1) \cdots \exp(A_q) = F_k$. If $|F_k| = 1$, we proceed as in Section 3. Otherwise, we first call `MergeEnclosed`($p, p + \ell - 1$), where $\ell = |F_k|$ and $T[p..p + \ell)$ is the source of F_k . The sequence A_1, \dots, A_q is then obtained as a result of `DecomposeWithRoots`($p, p + \ell - 1$). Finally, A_1, \dots, A_q , is appended to the roots sequence.

The above algorithm runs in $\mathcal{O}(f \log^2 n)$ time. To see this, note that first calling `MergeEnclosed` ensures that the output size of `DecomposeWithRoots` is $\mathcal{O}(\log n)$. Thus, each step appends only $\mathcal{O}(\log n)$ nonterminals to the roots sequence. The total time spend in `MergeEnclosed` is thus bounded by $\mathcal{O}(f \log^2 n)$, dominating the time complexity.

To prove the correctness of the modified algorithm, we need to prove that: (1) every nonterminal in the new algorithm satisfies the AVL property, and (2) after iteration $k \in [1..f]$, the invariant $\exp(R_1) \cdots \exp(R_m) = F_1 \cdots F_k$ holds. To show (1), we note that in the above algorithm, the nonterminals are only created by the `MergeEnclosed` procedure. Internally, this procedure calls `AddMerged`, which guarantees that the newly created nonterminal satisfies the AVL property (see Section 3). To show (2), we first note that, by definition, `MergeEnclosed` does not change $\exp(R_1) \cdots \exp(R_m)$ (although it may change m). The expansion of the roots sequence changes only after appending the sequence of nonterminals A_1, \dots, A_q returned by `DecomposeWithRoots`($p, p + \ell - 1$). Since $T[p..p + \ell)$ is the source of F_k , we thus have $\exp(A_1) \cdots \exp(A_q) = (F_1 \cdots F_{k-1})[p..p + \ell) = T[p..p + \ell) = F_k$.

Utilizing Karp–Rabin Fingerprints. Our second technique is designed to detect the situation in which the algorithm adds a nonterminal A to G , when there already exists some $B \in N$ such that $\exp(A) = \exp(B)$. For any $u \in (N \cup \Sigma)^*$, we define $\Phi(u) = \Phi(\exp(u))$. Let us assume that there are no collisions between fingerprints.² During the algorithm, we maintain a collection of fingerprints $\{\Phi(A) : A \in N'\}$, where $N' \subseteq N$ is some subset of nonterminals. Assume, that given a nonterminal $A \in N$, we can quickly compute $\Phi(A)$, and that given some $x \geq 0$, we can check, if there exists $B \in N'$ such that $\Phi(B) = x$. There are two places in the above algorithm (using lazy merging) where we utilize this to reduce the number of nonterminals:

1. Whenever during the greedy merge in `MergeEnclosed`, we are about to call `AddMerged` for the pair of adjacent nonterminals X and Y , we instead first compute the fingerprint $x = \Phi(XY)$ of their concatenation, and if there already exists $A \in N'$ such that $\Phi(A) = x$, we use A instead, avoiding the call to `AddMerged` and the creation of extra nonterminals.
2. Before appending the nonterminals A_1, \dots, A_q to the roots sequence at the end of the step, we check if there exists an equivalent but shorter sequence $B_1, \dots, B_{q'}$, i.e., such that $\exp(A_1) \cdots \exp(A_q) = \exp(B_1) \cdots \exp(B_{q'})$ and $q' < q$. We utilize that $q = \mathcal{O}(\log n)$, and run a quadratic algorithm (based on dynamic programming) to find the optimal (shortest) equivalent sequence. We then use that equivalent sequence in place of A_1, \dots, A_q .

Observe that the above techniques cannot be applied during `AddMerged`, as the equivalent nonterminal could have a much shorter/taller parse tree, violating the AVL property. Note also that the size and contents of N' do not affect the correctness or the time complexity of the algorithm. To determine the set N' , our implementation uses a parameter $p \in [0..1]$, which is the probability of adding A to N' , whenever a new nonterminal A is created. The value of

² Although such an assumption can be ensured with probability $1 - n^{-c}$ for any constant $c > 0$, it cannot be easily guaranteed. This turns our algorithm into a Monte Carlo randomized algorithm.

p is one of the main parameters controlling the time-space trade-off of the algorithm, as well as the size of the final grammar. To check if there exists $A \in N'$ such that $\Phi(A) = x$, for a given $x \geq 0$, we maintain a hash table that maps the values from the set $\{\Phi(A) : A \in N'\}$ to the corresponding nonterminals (each nonterminal is assigned a unique integer identifier).

5 Implementation Details

Storing Sequences. Our implementation stores many sequences, where the insertion only happens at the end (e.g., the sequence of nonterminals, which are never deleted in the algorithm). A standard approach to this is to use a *dynamic array*, which is a plain array that doubles its capacity, once it gets full. Such implementation achieves an amortized $\mathcal{O}(1)$ insertion time, but suffers from a high peak RAM usage. On all systems we tried, the reallocation call that extends the array is not in-place. Since the peak RAM usage is critical in our implementation, we implemented our own dynamic array that instead of a single allocated block, keeps a larger number of blocks (we use 32). This significantly reduces the peak RAM usage. We found the slowdown in the access-time to be negligible.

Implementation of the Roots Sequence. The roots sequence $(\ell_1, R_1), \dots, (\ell_m, R_m)$ undergoes predecessor queries, deletions (at arbitrary positions), and insertions (only at the end). Rather than using an off-the-shelf dynamic predecessor data structure (such as balanced BST), we exploit as follows the fact that insertions happen only at the end.

All roots are stored as a static sequence that only undergoes insertions at the end (using the space efficient dynamic array implementation described above). Deleted elements are marked as deleted, but remain physically in the array. The predecessor query is implemented using a binary search, with skipping of the elements marked as deleted. To ensure that the predecessor queries are efficient, we keep a counter of accesses to the deleted elements. Once it reaches the current array size, we run the “garbage collector” that scans the whole array left-to-right, and removes all the elements marked as deleted, eliminating all gaps. This way, the predecessor query is still efficient, except the complexity becomes amortized.

Computing $\Phi(A)$ and $|\text{exp}(A)|$ for $A \in N$. During the algorithm, we often need to query the value of $\Phi(A)$ for some nonterminal $A \in N$. In our implementation we utilize 64-bit fingerprints, and hence storing the value $\Phi(A)$ for every nonterminal is expensive. We thus only store $\Phi(A)$ for $A \in N$ satisfying $|\text{exp}(A)| \geq 255$. The number of such elements in N is relatively small. To compute $\Phi(A)$ for any other $A \in N$, we first obtain $\text{exp}(A)$, and then compute $\Phi(A)$ from scratch. This operation is one of the most expensive in our algorithm, and hence whenever possible we avoid doing repeated Φ queries.

As for the values $|\text{exp}(A)|$, we observe that in most cases, it fits in a single byte. Thus, we designate only a single byte, and whenever $|\text{exp}(A)| \geq 255$, we lookup $|\text{exp}(A)|$ in an array ordered by the number of nonterminal, with access implemented using binary search.

6 Experimental Results

Algorithms. We performed experiments using the following algorithms:

- Basic-AVLG, our implementation of the algorithm to convert an LZ-like parsing to an AVL grammar proposed by Rytter [30], and outlined in Section 3. Self-referential phrases are handled as in the full version of [20, Theorem 6.1]. The implementation uses space-efficient dynamic arrays described in Section 5. This implementation is our baseline.

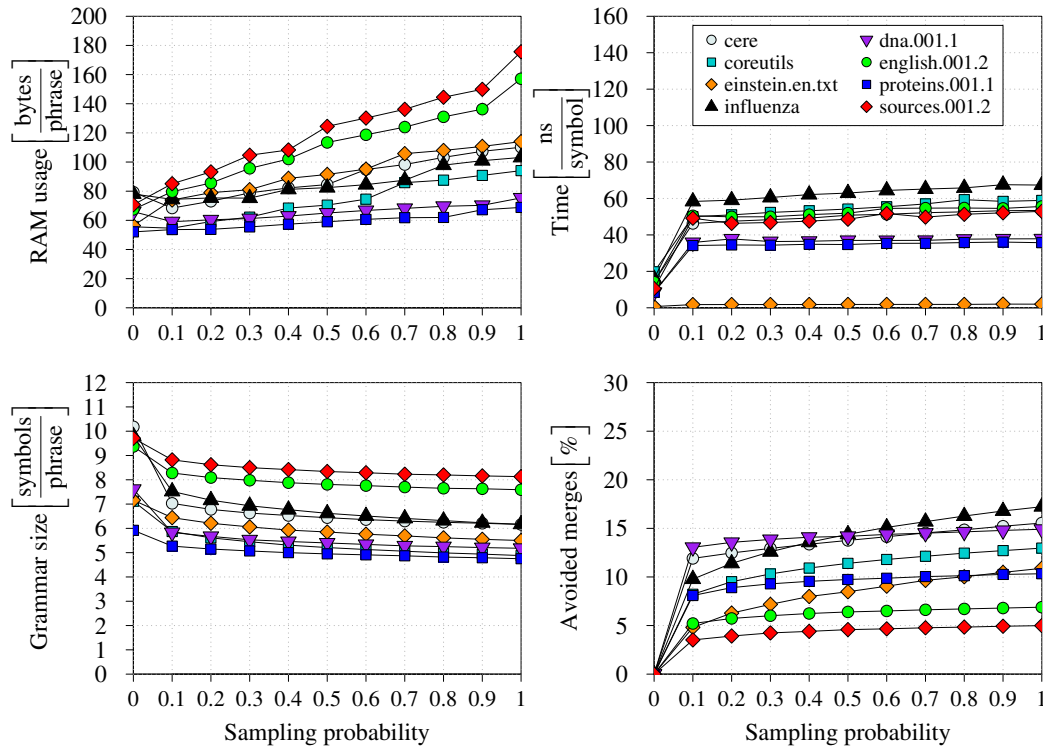
■ **Table 1** Statistics of files used in the experiments, with n denoting text length, σ denoting alphabet size, r denoting the number of runs in the BWT, and z denoting the number of phrases in the LZ77 parsing. For convenience, we also show the average BWT run length n/r and the average LZ77 phrase length n/z . Each of the symbols in the input texts is encoded using a single byte.

File name	n	σ	r	n/r	z	n/z
cere	461 286 644	5	11 574 640	39.85	1 700 630	271.24
coreutils	205 281 778	236	4 684 459	43.82	1 446 468	141.91
einstein.en.txt	467 626 544	139	290 238	1611.18	89 467	5 226.80
influenza	154 808 555	15	3 022 821	51.21	769 286	201.23
dna.001.1	104 857 600	5	1 716 807	61.07	308 355	340.05
english.001.2	104 857 600	106	1 449 518	72.33	335 815	312.24
proteins.001.1	104 857 600	21	1 278 200	82.03	355 268	295.15
sources.001.2	104 857 600	98	1 213 427	86.41	294 994	355.45
chr19.1000	59 125 116 167	5	45 927 063	1287.37	7 423 960	7964.09
kernel	137 438 953 472	229	129 506 377	1061.25	30 222 602	4547.55

- Lazy-AVLG, our implementation of the improved version of Basic-AVLG, utilizing lazy merging and Karp–Rabin fingerprints, as described in Section 4. This is the main contribution of our paper. In some of the experiments below, we consider the algorithm with the different probability p of sampling the Karp–Rabin hash of a nonterminal, but our default value (as discussed below) is $p = 0.125$. Our implementation (including also Basic-AVLG) is available at <https://github.com/dominikkempa/lz77-to-slp>.
- Big-BWT, a semi-external algorithm constructing the RLBWT from the input text in $\Omega(n)$ time, proposed by Boucher et al. [4]. As shown in [4], if the input text is highly compressible, the working space of Big-BWT is sublinear in the text length. We use the implementation from <https://gitlab.com/manzai/Big-BWT>.
- Re-LZ, an external-memory algorithm due to Kosolobov et al. [24] that given a text on disk, constructs its LZ-like parsing in $\Omega(n)$ time [24]. The algorithm is faster than the currently best algorithms to compute the LZ77 parsing. In practice, the ratio f/z between the size f of the resulting parsing and the size z of the LZ77 parsing usually does not exceed 1.5. Its working space is fully tunable and can be specified arbitrarily. We use the implementation from <https://gitlab.com/dvalenzu/ReLZ>.
- Re-Pair, an $\mathcal{O}(n)$ -time algorithm to construct an SLG from the text, proposed by Larsson and Moffat [25]. Although no upper bound is known on its output size, Re-Pair produces grammars that in practice are smaller than any other grammar compression method [25]. Its main drawback is that most implementations need $\Theta(n)$ space [11], and hence are not applicable on massive datasets. The only implementation using $o(n)$ space is [23], but as authors note themselves, it is not practical. There is also work on running Re-Pair on the compressed input [31], but since it already requires the text as a grammar, it is not applicable in our case. In our experiments we therefore use Re-Pair only as a baseline for the achievable grammar size. We note that there exists recent work on optimizing Re-Pair by utilizing maximal repeats [11]. The decrease in the grammar size, however, requires a potentially more expensive nonterminal encoding that includes the length of the expansion. For simplicity, we therefore use the basic version of Re-Pair.

All implementations are in C++ and are largely sequential, allowing for a constant number of additional threads used for asynchronous I/O.

We also considered Online-RLBWT, an algorithm proposed by Ohno et al. [28], that given a text in a right-to-left streaming fashion, construct its run-length compressed BWT (RLBWT) in $\mathcal{O}(n \log r)$ time and using only $\mathcal{O}(r \log n)$ bits of working space (the implementa-



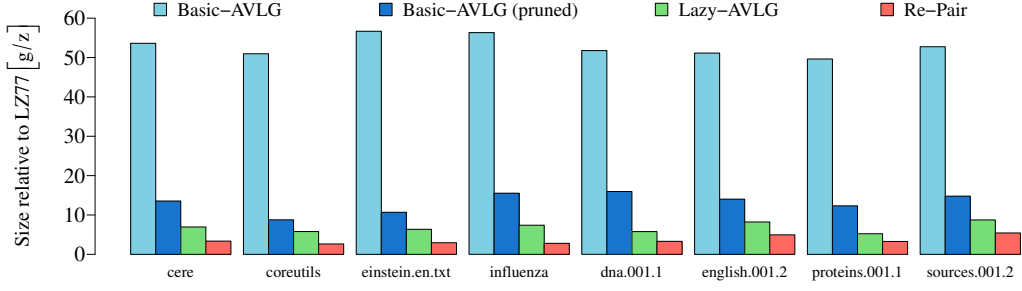
■ **Figure 1** Performance of the Lazy-AVLG algorithm for different values of the parameter p (probability of storing Karp–Rabin fingerprint in the hash table) on the files from Pizza & Chili corpus. The graphs in the top row show the normalized RAM usage (in bytes per phrase of the LZ77 parsing) and the runtime (in ns per symbols of the input text). The bottom row shows the resulting grammar size (the total length of right-hand sides of all productions) divided by z , and the percentage of merges avoided during the greedy merge procedure (in %).

tion is available from: <https://github.com/itomomoti/OnlineRlbwt>). In the preliminary experiment we determined that while using only about a third of the memory of Big-BWT (on the 16 GiB prefix of the kernel testfile), the algorithm was about 10x slower than Big-BWT. We also did not include [14], since in the preliminary experiments we found it to be slower (usually by about 5–10%) than Big-BWT, while using about 1.8x more RAM.

Experimental Platform and Datasets. We performed experiments on a machine equipped with two twelve-core 2.2 GHz Intel Xeon E5-2650v4 CPUs with 30 MiB L3 cache and 512 GiB of RAM. The machine used distributed storage achieving an I/O rate >220 MiB/s (read/write).

The OS was Linux (CentOS 7.7, 64bit) running kernel 3.10.0. All programs were compiled using g++ version 4.8.5 with `-O3 -DNDEBUG -march=native` options. All reported runtimes are wallclock (real) times. The machine had no other significant CPU tasks running. To measure the peak RAM usage of the programs we used the `/usr/bin/time -v` command.

The statistics of testfiles used in our experiments are shown in Table 1. Shorter version of files used in the scalability experiments are prefixes of full files. We used the files from the Pizza & Chili repetitive corpus available at <http://pizzachili.dcc.uchile.cl/repcorpus.html>. We chose a sample of 8 real and pseudo-real files. Since all files are relatively small (less than 512 MiB), we additionally include 2 large repetitive files:

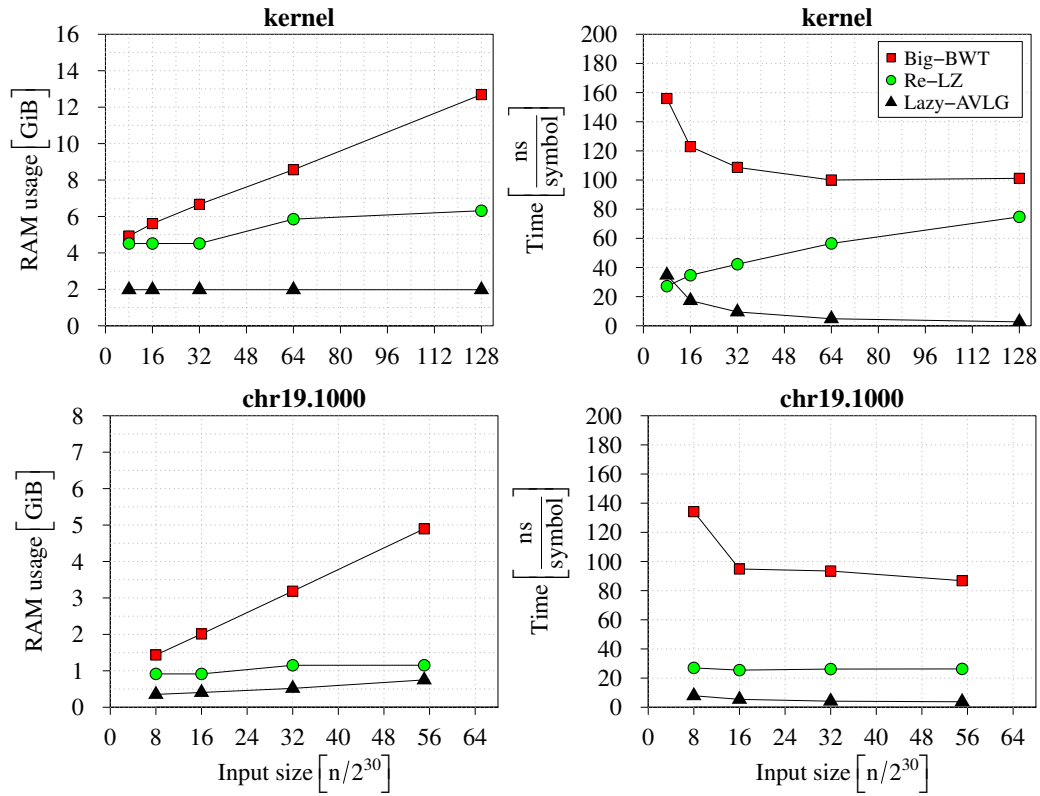


■ **Figure 2** Comparison of the size (measured as the total length of the right hand sides of all nonterminals) of grammars produced by the Basic-AVLG, Lazy-AVLG ($p = 0.125$), and Re-Pair algorithms on the files from the Pizza & Chili corpus. Basic-AVLG (pruned) denotes the size of grammar produced by Basic-AVLG with all nonterminals not reachable from the root removed. All sizes are normalized with respect to the size of the LZ77 parsing (z).

- chr19.1000, a concatenation of 1000 versions of Human chromosome 19. The sequences were obtained from the 1000 Genomes Project [8]. One copy consists of $\sim 58 \times 10^6$ symbols.
- kernel, a concatenation of ~ 10.7 million source files from over 300 versions of the Linux kernel (see <http://www.kernel.org/>).

Karp–Rabin Sampling Rate. The key parameter that controls the runtime, peak RAM usage, and the size of the grammar in our algorithm is the probability p of including the Karp–Rabin fingerprint of a nonterminal in the hash table. In our first experiment, we study the performance of the Lazy-AVLG algorithm for different values of the parameter p . We tested all values $p \in \{0, 0.1, 0.2, \dots, 1\}$ and for each we measured the algorithm’s time and memory usage, and the size of the final grammar. The results are given in Figure 1.

While utilizing the Karp–Rabin fingerprints (i.e., setting $p > 0$) can notably reduce the final grammar size (up to 40% for the *cere* file), it is not worth using values p much larger than 0.1, as it quickly increases the peak RAM usage (e.g., by about 2.3x for the *sources.001.2* testfile) and this increase is not repaid significantly in the further grammar reduction. The bottom right panel in Figure 1 provides some insight into the reason for this. It shows the percentage of cases, where during the greedy merging of nonterminals enclosed by the source of the phrase, the algorithm is able to avoid merging two nonterminals, and instead use the existing nonterminal. Having *some* fingerprints in the hash table turns out to be enough to avoid creating between 4–14% of the new nonterminals, but having more does not lead to a significant difference. We also observe that if a larger grammar is acceptable, disabling the use of Karp–Rabin fingerprints entirely (i.e., setting $p = 0$) can lead to a significant speed-up (the top right panel in Figure 1) and a small saving in the RAM usage (note, that it also makes the algorithm deterministic). We choose to use $p > 0$, however, since in our main application (BWT construction), the final grammar is subject to further processing, and since this processing may dominate the RAM usage, we prefer to keep the grammar as small as possible. Since peak RAM usage is the likely limiting factor for this algorithm – a slower algorithm is still usable, but exhaustion of RAM can prevent it running entirely – we chose $p = 0.125$ as the default value in our implementation (and use in the next two experiments).



■ **Figure 3** Scalability of Big-BWT compared to Re-LZ and Lazy-AVLG. The graphs on the right show the normalized runtime in ns/char. The graphs on the left show the RAM usage in GiB.

Grammar Size. In our second experiment, we compare the size of the grammar produced by Lazy-AVLG to Basic-AVLG and Re-Pair. In the comparison we also include the size of the grammar obtained by running Basic-AVLG and removing all nonterminals not reachable from the root. We have run the experiments on 8/10 testfiles, as running Re-Pair on the large files is prohibitively time consuming. The results are given in Figure 2.

Lazy-AVLG produces grammars that are between 1.59x and 2.64x larger than Re-Pair (1.95x on average). The resulting grammar is always at least 5x smaller than produced by Basic-AVLG, and also always smaller than Basic-AVLG (pruned). Importantly, the RAM usage of our conversion is proportional the size of the final grammar, whereas the algorithm to compute the pruned version of Basic-AVLG must first obtain the initial large grammar, increasing peak RAM usage. This is a major practical concern, as described in the next experiment. In conclusion, Lazy-AVLG compresses only slightly worse than Re-Pair, but its construction requires much less working space.

Application in the Construction of BWT. In our third experiment, we evaluate the potential of the method to construct the RLBWT presented in [20], which works by first computing/approximating the LZ77 parsing of the text in $\Omega(n)$ time, and then converting the resulting compressed representation of T into the RLBWT in $\mathcal{O}(f \text{ polylog } n)$ time (where f denotes the number of factors). We use Re-LZ to implement the first step. As for the second step, we note that the conversion from the (approximate) LZ77 to RLBWT internally consists of two steps: (2a) (approximate) LZ77 \rightarrow grammar, and (2b) grammar \rightarrow RLBWT.

In this experiment, we use Lazy-AVLG to implement step (2a). We have not implemented the step (2b), and leave it as a future work. The results reported here are therefore only a preliminary indication of what is achievable with the approach of [20]. Our baseline for the construction of RLBWT is the Big-BWT algorithm [4].

We evaluated the runtime and peak RAM usage of Big-BWT, Re-LZ, and Lazy-AVLG with $p = 0.125$ (the default value) on successively longer prefixes of the large testfiles (chr19.1000 and kernel). The RAM use of Re-LZ was set to match the RAM use of Big-BWT on the shortest prefix we tried. To allow a comparison with different methods in the future, we evaluated Lazy-AVLG on the LZ77 parsing rather than on the output of Re-LZ. Thus, to obtain the performance of the pipeline Re-LZ + Lazy-AVLG, one should multiply the runtime and RAM usage of Lazy-AVLG by the approximation ratio f/z of Re-LZ. The value f/z did not exceed 1.05 on any of the kernel prefixes, and 1.27 on any of the chr19.1000 prefixes (with the peak reached on the largest prefixes). This puts the RAM use of Lazy-AVLG on the output of Re-LZ still below that of Re-LZ. The results are given in Figure 3.

The runtime of Re-LZ is always below that of Big-BWT. The reduction is by a factor of at least three for all prefixes of chr19.1000, and by at least 25% for all prefixes of kernel. The runtime of Lazy-AVLG stays significantly below that of both other methods. Importantly, this also holds for Lazy-AVLG's peak RAM usage. Given these results, we conclude that the construction of the RLBWT via LZ parsing has the potential to achieve at least a three-fold speedup and reduction in the RAM usage. We also point out that the construction of RLBWT has received much attention in recent years, whereas the practical approximation of LZ77 is a relatively unexplored topic, and hence significant speedup may be possible, e.g., via parallelization. The intuition for this is that, unlike in the case of BWT construction, LZ approximation algorithms need not be exact.

References

- 1 Djamal Belazzougui, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *DCC*, pages 83–92. IEEE, 2015. doi:10.1109/DCC.2015.69.
- 2 Timo Bingmann, Johannes Fischer, and Vitaly Osipov. Inducing suffix and LCP arrays in external memory. In *ALENEX*, pages 88–102. SIAM, 2013. doi:10.1137/1.9781611972931.8.
- 3 Christina Boucher, Travis Gagie, Tomohiro I, Dominik Köppl, Ben Langmead, Giovanni Manzini, Gonzalo Navarro, Alejandro Pacheco, and Massimiliano Rossi. PHONI: Streamed matching statistics with multi-genome references. In *DCC*, pages 193–202. IEEE, 2021. doi:10.1109/DCC50243.2021.00027.
- 4 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms Mol. Biol.*, 14(1):13:1–13:15, 2019. doi:10.1186/s13015-019-0148-5.
- 5 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
- 6 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
- 7 Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundam. Informaticae*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.
- 8 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015. doi:10.1038/nature15393.
- 9 Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable. In *ICALP*, pages 235–246, 1992. doi:10.1007/3-540-55719-9_77.

- 10 Paolo Ferragina, Travis Gagie, and Giovanni Manzini. Lightweight data indexing and compression in external memory. *Algorithmica*, 63(3):707–730, 2012.
- 11 Isamu Furuya, Takuya Takagi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Takuya Kida. MR-RePair: Grammar compression based on maximal repeats. In *DCC*, pages 508–517. IEEE, 2019. doi:10.1109/DCC.2019.00059.
- 12 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *LATIN*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.
- 13 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *LATA*, pages 240–251. Springer, 2012. doi:10.1007/978-3-642-28332-1_21.
- 14 Travis Gagie, Tomohiro I, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, and Yoshimasa Takabatake. Rpair: Rescaling RePair with Rsync. In *SPIRE*, pages 35–44, 2019. doi:10.1007/978-3-030-32686-9_3.
- 15 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In *LATIN*, pages 490–503. Springer, 2018. doi:10.1007/978-3-319-77404-6_36.
- 16 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):1–54, April 2020. doi:10.1145/3375890.
- 17 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lempel-Ziv parsing in external memory. In *DCC*, pages 153–162. IEEE, 2014. doi:10.1109/DCC.2014.78.
- 18 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Parallel external memory suffix sorting. In *CPM*, pages 329–342. Springer, 2015. doi:10.1007/978-3-319-19929-0_28.
- 19 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.
- 20 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler Transform conjecture. In *FOCS*, pages 1002–1013. IEEE, 2020. Full version: <https://arxiv.org/abs/1910.10631>. doi:10.1109/FOCS46700.2020.00097.
- 21 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *STOC*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.
- 22 Donald E. Knuth. *The Art of Computing, Vol. III, 2nd Ed.* Addison-Wesley, 1998.
- 23 Dominik Köppl, Tomohiro I, Isamu Furuya, Yoshimasa Takabatake, Kensuke Sakai, and Keisuke Goto. Re-Pair in small space. *Algorithms*, 14(1):5, 2021. doi:10.3390/a14010005.
- 24 Dmitry Kosolobov, Daniel Valenzuela, Gonzalo Navarro, and Simon J. Puglisi. Lempel-Ziv-like parsing in small space. *Algorithmica*, 82(11):3195–3215, 2020. doi:10.1007/s00453-020-00722-6.
- 25 N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.
- 26 Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.
- 27 Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surv.*, 54(2):26:1–26:32, 2021. doi:10.1145/3432999.
- 28 Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A faster implementation of online RLBWT and its application to LZ77 parsing. *J. Discrete Alg.*, 52-53:18–28, 2018. doi:10.1016/j.jda.2018.11.002.
- 29 Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*, 80(7):1986–2011, 2018. doi:10.1007/s00453-017-0327-z.
- 30 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

- 31 Kensuke Sakai, Tatsuya Ohno, Keisuke Goto, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. RePair in compressed space and time. In *DCC*, pages 518–527. IEEE, 2019. doi:10.1109/DCC.2019.00060.
- 32 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

Convex Drawings of Hierarchical Graphs in Linear Time, with Applications to Planar Graph Morphing*

Boris Klemz 

Universität Würzburg, Germany

Abstract

A hierarchical plane st-graph H can be thought of as a combinatorial description of a planar drawing Γ of a 2-connected graph G in which each edge is a y -monotone curve and each face encloses a y -monotone region (that is, a region whose intersection with any horizontal line is a line segment, a point, or empty). A drawing Γ' of H is a drawing of G such that each horizontal line intersects the same left-to-right order of edges and vertices in Γ and Γ' , that is, the underlying hierarchical plane st-graph of both drawings is H . A straight-line planar drawing of a graph is convex if the boundary of each face is realized as a convex polygon.

We study the computation of convex drawings of hierarchical plane st-graphs such that the outer face is realized as a prescribed polygon. Chrobak, Goodrich, and Tamassia [SoCG'96] and, independently, Kleist et al. [CGTA'19] described an idea to solve this problem in $\mathcal{O}(n^{1.1865})$ time, where n is the number of vertices of the graph. Also independently, Hong and Nagamochi [J. Discrete Algorithms'10] described a completely different approach, which can be executed in $\mathcal{O}(n^2)$ time.

In this paper, we present an optimal $\mathcal{O}(n)$ time algorithm to solve the above problem, thereby improving the previous results by Chrobak, Goodrich, and Tamassia, Kleist et al., and by Hong and Nagamochi. Our result has applications in graph morphing. A planar morph is a continuous deformation of a graph drawing that preserves straight-line planarity. We show that our algorithm can be used as a drop-in replacement to speed up a procedure by Alamdari et al. [SICOMP'17] to morph between any two given straight-line planar drawings of the same plane graph. The running time improves from $\mathcal{O}(n^{2.1865})$ to $\mathcal{O}(n^2 \log n)$. To obtain our results, we devise a new strategy for computing so-called archfree paths in hierarchical plane st-graphs, which might be of independent interest.

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 convex drawing, hierarchical graph, graph drawing, computational geometry, planarity, planar graph, morphing, convexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.57

Funding Boris Klemz: supported by DFG project WO 758/11-1.

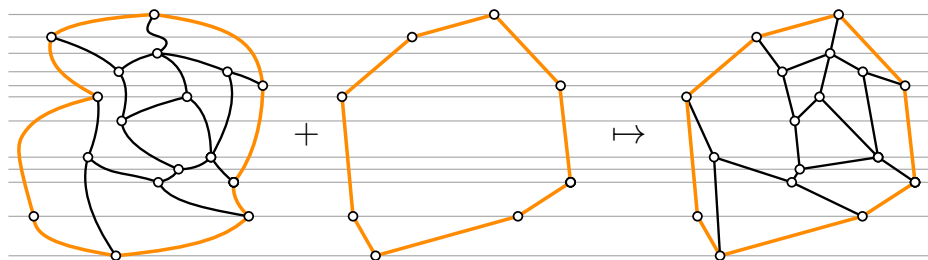


Figure 1 A y -monotone drawing (left) and a convex drawing of its underlying hierarchical plane st-graph (right) with a prescribed compatible polygon (middle) as the outer face.

* Due to space constraints, some proofs in this extended abstract are only sketched or omitted entirely.



1 Introduction

A *y-monotone* drawing Γ is a planar drawing of a 2-connected planar graph $G = (V, E)$ where each edge is realized as a *y-monotone* curve and the boundary of each face encloses a *y-monotone* region, that is, a region whose intersection with any horizontal line is a line segment, a point, or empty; for an illustration see Figure 1. Due to the *y-monotone* edges, the drawing Γ uniquely determines for each $v \in V$ (1) the *y*-coordinate $y(v)$ of v , (2) a left-to-right ordering of the edges incident to v that have an endpoint with a *y*-coordinate larger than $y(v)$, and (3) a left-to-right ordering of the edges incident to v that have an endpoint with a *y*-coordinate smaller than $y(v)$. A plane graph is a combinatorial description of a planar drawing of a graph that consists of the graph equipped with the so-called combinatorial embedding of the drawing and a distinguished outer face. Similarly, the underlying *hierarchical plane st-graph* H of Γ is a combinatorial description of the *y-monotone* drawing Γ that consists of G equipped with the above information (Items (1)–(3)) for each vertex. A *drawing* Γ' of H is a *y-monotone* drawing of G whose underlying hierarchical plane st-graph is H . A planar straight-line drawing of a graph is called *convex* if the boundary of each face is realized as a simple convex polygon.

This paper is concerned with the computation of convex drawings of hierarchical plane st-graphs such that the outer face is realized as a prescribed polygon that is, in some sense, compatible with the given graph; for an illustration see Figure 1. A plane graph admits a convex drawing if and only if it is a subdivision of an internally 3-connected graph. Hence, we will assume that our input graphs satisfy this property.

The above problem has applications in graph morphing, as we will discuss next.

Applications. A (*planar*) *morph* is a continuous deformation of a graph drawing that preserves straight-line edges (and planarity) at all times. Graph morphing is motivated by applications in animation and computer graphics [17]. In computational morphing problems, one typically seeks “piece-wise linear” morphs, which are composed of a number of linear morphing steps. In a *linear* morph, each vertex moves along a line segment at constant speed (which depends on the length of the segment) such that it arrives at its final position at the end of the morph. Such a morph is uniquely defined by specifying the initial and the final drawing. Hence, a morph composed of k linear morphs can be efficiently encoded as a sequence of $k + 1$ drawings.

Algorithms to compute convex drawings of hierarchical plane st-graphs such that the outer face is realized as a prescribed polygon serve as a subroutine in several graph morphing algorithms [2, 22, 7, 27, 6]. The key observation is that the linear morph from a *y-monotone* straight-line drawing to a convex drawing of its underlying hierarchical plane st-graph is planar, which is not difficult to prove due to the fact that its vertex trajectories are parallel lines [2]; a linear morph with this property is called *unidirectional*. The restriction to *y-monotone* drawings might seem limiting at first. However, the observation is also useful in a more general context: let Γ be a straight-line planar drawing. Let Γ' be a *y-monotone* augmentation of Γ created by adding a set A of *y-monotone* (but not necessarily straight-line) edges. Finally, let Γ'' be a convex drawing of the underlying hierarchical plane st-graph of Γ' . Then the linear morph from Γ to $\Gamma'' \setminus A$ is also planar [22]. Note that all angles of face boundaries in $\Gamma'' \setminus A$ that were not subdivided by edges of A are convex. In this sense, $\Gamma'' \setminus A$ is a simplified version of Γ , which can be exploited algorithmically. We will illustrate this by discussing an explicit example.

One of the most basic tasks in graph morphing is the computation of a planar morph (composed of linear morphing steps) between two given straight-line planar drawings Γ_1, Γ_2 of the same plane graph G . Alamdari et al. [2] described an algorithm to compute such a morph consisting of a linear number of unidirectional steps. They provide a reduction to show that it suffices to consider the case that Γ_1, Γ_2 , and G are triangulated. For triangulations, the idea is as follows: find a suitable edge $e = \{u, v\}$ shrink it to a point in both drawings, thereby contracting the edge e in G to a new vertex w . Then, recursively compute a morph \mathcal{M} of the reduced graph. Finally, turn the thereby obtained “pseudomorph” of G into an actual morph by placing u and v very close to the position of w in each of the drawings encoding \mathcal{M} . To find an edge e that can be contracted without violating planarity in either drawing, proceed as follows: let u be an internal vertex with $\deg_G(u) \leq 5$. If the polygon defined by the neighbors of u is convex in both Γ_1 and Γ_2 , then u can be contracted towards the same neighbor in both drawings. If this is not the case, the polygon (or some specific angle) can be made convex using the strategy described in the preceding paragraph.

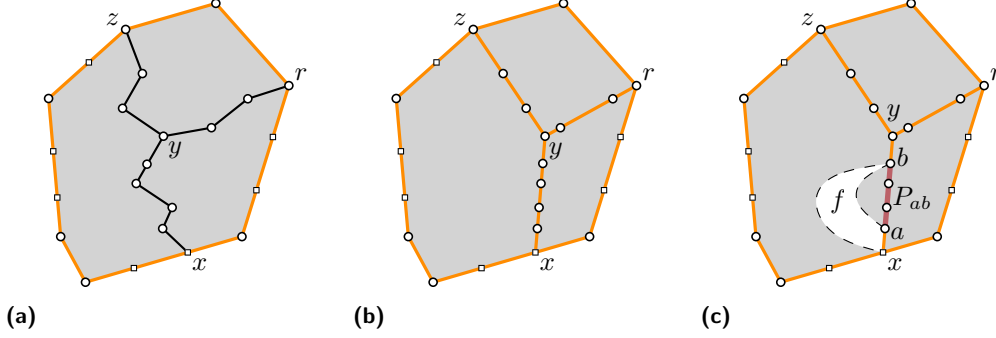
In other types of morphing problems, the task is to compute planar morphs between two given drawings while maintaining additional properties such as convexity [7] or upward-planarity [27], or planar morphs that transform a given drawing in order to achieve a certain property while being in some sense monotonic [1, 22, 11]. Very recently, the problem of morphing graphs was also studied on the torus [9].

We remark that the computation of convex drawings of hierarchical plane st-graphs with a prescribed polygon as the outer face also plays a role when embedding polyhedral graphs in \mathbb{R}^3 with a good vertex resolution [10, 29]. The idea is to first find a convex drawing in the plane where the vertices are placed at, suitably chosen, prescribed small integer y -coordinates. The drawing is then lifted to \mathbb{R}^3 . However, to ensure this strategy can be carried out, the constructed plane drawings are also required to be embeddings with so-called equilibrium stress – the drawings created by the algorithm presented in this paper do not guarantee this property.

Previous algorithms. An idea for constructing convex drawings of hierarchical plane st-graphs with a prescribed polygon as the outer face was described already in 1996 by Chrobak, Goodrich, and Tamassia [10, Section 3] (in the context of realizing polyhedral graphs in \mathbb{R}^3); also see [29, Section 4]. The approach was independently rediscovered by Kleist et al. [22] (in the context of a morphing problem) and is based on using Tutte’s well-known spring theorem [32, 15, 16]. The main idea is to precompute barycenter weights that force the vertices to lie at the prescribed y -coordinates before applying Tutte’s algorithm, which then finds suitable x -coordinates. (Notably, the way in which the barycenter weights are determined in [22] is quite different from the method used in [10] and [29].) Chrobak et al. point out that the approach can be implemented in $\mathcal{O}(n^{\omega/2} + n \log n) \subseteq \mathcal{O}(n^{1.1865})$ time by using the generalized nested dissection method [25, 26, 4], where n is the number of vertices of the graph and $\omega < 2.37286$ [3, 24] denotes the matrix multiplication exponent.

In 2010, Hong and Nagamochi [20] described a completely different approach, based on a recursive combinatorial construction. The runtime of their algorithm is $\mathcal{O}(n^2)$. The idea is to choose a suitable internal vertex y of the given graph G and compute three disjoint (except for y) paths from y to the outer face, see Figure 2a. These paths dissect G into three regions, which are then handled recursively. The outer face of each of the three regions is composed of two of the three paths, which are prescribed to be realized as straight-line segments, and a part of the original prescribed polygon, see Figure 2b. To ensure that the thereby described polygon can be extended to a convex drawing of the entire region, the computed paths need to be *archfree*, meaning that they are not *arched* by an internal face. A path P is arched by

a face f if P contains two distinct vertices a, b that belong to the boundary of f such that the subpath P_{ab} of P between a and b is not a subpath of the boundary of f , see Figure 2c. Indeed, such a path P cannot be realized as a straight-line segment in a convex drawing since in this case the interior of the segment ab has to be disjoint from the realization of f .



■ **Figure 2** (a–b) The idea of Hong and Nagamochi’s [20] construction. (c) The path $P = (x, \dots, y)$ is arched by a face f .

We remark that Chrobak et al.’s result [10] does not appear to be widely known: the morphing papers [2, 7, 27, 6, 22] only refer to the method by Hong and Nagamochi [20]. Moreover, Hong and Nagamochi [20] and Kleist et al. [22] were also unaware of its existence.

Other related work. Pach and Tóth [28] and Eades, Feng, Lin, and Nagamochi [14] studied the problem of finding (not necessarily convex) straight-line drawings of *hierarchical plane graphs*, which can be defined as hierarchical plane *st*-graphs, except that they describe (not necessarily *y*-monotone) planar drawings in which each edge is realized as a *y*-monotone curve. Eades et al. [14] provide a linear-time algorithm for this problem that realizes the outer face as a prescribed polygon.

A graph in which each vertex is equipped with a *y*-coordinate is sometimes called a *level graph*. The central question in the LEVEL PLANARITY [13, 18, 21] problem and its many variants [5, 8, 23] is to decide whether a given level graph admits a *level planar drawing*, that is, a planar drawing in which each vertex is placed at its prescribed *y*-coordinate and each edge is realized as a *y*-monotone curve. By definition, the underlying level graph of each hierarchical plane (st-)graph admits a level planar drawing.

Contribution and organization. In Section 4, we describe an optimal linear-time algorithm for constructing convex drawings of hierarchical plane st-graphs with a prescribed polygon as the outer face (if possible), thereby improving the previous approaches by Chrobak, Goodrich, and Tamassia [10], Kleist et al. [22], and Hong and Nagamochi [20].

► **Theorem 1.** *There exists an algorithm that, given a subdivision $G = (V, E)$ of an internally 3-connected hierarchical plane st-graph and a convex polygon Γ^o that is compatible with G , computes a convex drawing of G with Γ^o as the realization of the outer face in time $\mathcal{O}(n)$ where $n = |V|$.*

We introduce our notation and terminology (including the definitions of compatible polygons and internally 3-connected graphs) and the assumed data structures in Section 2.

To obtain Theorem 1, we follow the idea of the recursive combinatorial construction by Hong and Nagamochi [20]. We observe that the main bottleneck in Hong and Nagamochi’s algorithm is the computation of archfree paths to the boundary: Hong and Nagamochi obtain

such a path by computing an arbitrary y -monotone path P which is then modified to obtain an archfree path P' . In general, not all vertices of P belong to P' . Hence, a given vertex may be involved in the archfree path computation on multiple layers of the recursion. Since the recursion depth can be linear, the overall runtime of their algorithm is quadratic. In Section 3, we describe a new efficient algorithm to obtain archfree paths in a more direct way: the running time of our approach is linear in the sum of the degrees of the internal vertices visited by the computed path. With this tool at hand, the layers of the recursion become, in a sense, disjoint (since archfree paths computed on distinct layers of the recursion are disjoint). This lets us carry out the algorithm corresponding to Theorem 1 in linear time.

► **Theorem 2.** *There exists an algorithm that, given an internally 3-connected hierarchical plane st-graph G and (a pointer to) an internal vertex y of G , computes an archfree directed path P_{yz} from y to some outer vertex z of G such that all vertices of P_{yz} except for z are internal vertices of G in time $\mathcal{O}(\sum_{i \in V(P_{yz}) \setminus \{z\}} \deg_G^+(i))$.*

Alamdari et al. state in [2] that their morphing algorithm, which was already mentioned above, can be executed in time $\mathcal{O}(n^3)$. It uses Hong and Nagamochi's [20] algorithm for constructing convex drawings of hierarchical plane st-graphs as a blackbox with running time $\mathcal{O}(n^2)$. Kleist et al. [22] observed that by replacing the contents of this blackbox with an algorithm based on Tutte's theorem (as described by Chrobak et al. [10]), the running time can be improved to $\mathcal{O}(n^{1+\omega/2} + n^2 \log n) \subseteq \mathcal{O}(n^{2.1865})$. By plugging in Theorem 1 instead, we further improve the running time to $\mathcal{O}(n^2 \log n)$. In fact, if the graph is 2-connected, the running time can be improved to $\mathcal{O}(n^2)$. [2, Theorem 1.1] and [22, Theorem 2] become:

► **Theorem 3.** *There exists an algorithm that, given two straight-line planar drawings of the same n -vertex plane graph G , computes a planar morph between the two drawings that consists of $\mathcal{O}(n)$ unidirectional morphs in time $\mathcal{O}(n^2 \log n)$, and in time $\mathcal{O}(n^2)$ if G is 2-connected.*

Alamdari et al. [2] proved that the number of linear morphing steps to morph between two planar straight-line drawings of a path is bounded by $\Omega(n)$. This bound easily extends to the 2-connected case [22, Theorem 3]. Recall that a linear morph is uniquely defined by the initial and the final drawing. Hence, a natural way to encode a morph composed of k linear morphing steps is to provide a list of $k + 1$ drawings. Since the size of each drawing is $\Omega(n)$, the $\Omega(n)$ bound for the number of morphing steps implies an output complexity of $\Omega(n^2)$ when the morph is assumed to be encoded in this fashion. Hence, in this model, Theorem 3 is near-optimal, and optimal in the 2-connected case. It is an interesting question whether allowing some sort of implicit encoding can lead to better running times.

It seems likely that a similar running time improvement can be obtained for other morphing algorithms (such as the ones stated in [22, 7, 27]), though, we have not analyzed the respective running times in detail yet. We believe that Theorem 3 might be a useful tool for designing morphing algorithms in the future.

2 Terminology, data structures, and preliminary results

All graphs in this paper are simple, that is, we do not allow parallel edges or self-loops. Let $G = (V, E)$ be a graph. We denote by $V(G) = V$ the vertex set and by $E(G) = E$ the edge set of G . Assume that G is planar and let Γ be a planar drawing of G . The boundary of each face f of G can be uniquely described by a counterclockwise sequence of edges, or multiple such sequences if G is disconnected. In the connected case, we use ∂f to denote the boundary of f . If G is 2-connected, then ∂f is a *simple* cycle (otherwise, ∂f can visit vertices

and edges multiple times). The drawing Γ determines a circular ordering of the neighbors of each vertex. The set of these orderings together with the set of face boundaries is called the *combinatorial embedding* of Γ . Two drawings of G may have the same combinatorial embedding, but different outer faces. A *plane* graph is a planar graph equipped with a combinatorial embedding and a distinguished outer face. A plane graph can be efficiently encoded and traversed by means of the well-known *doubly connected edge list* (DCEL) [12].

Let H be a hierarchical plane st-graph. We consider each edge of H to be directed from its lower to its higher endpoint. Note that H has a unique source and sink, which belong to the outer face. Moreover, the boundary of each face of H has a unique source and sink. We refer to this as the *st-property* of H . For a face f of H , we define its *peak*, denoted by $\text{peak}(f)$, to be the y -coordinate of its sink. The *valley* of f , denoted by $\text{valley}(f)$, is the y -coordinate of its source. Let $v \in V(H)$ such that v is not the source or sink of H . There is a natural partition of the faces incident to v : a face with two out-edges of v on its boundary is called an *up-face* of v . Similarly, a face with two in-edges of v on its boundary is called a *down-face* of v . The unique *left-face* of v has the left-most out-edge and the left-most in-edge of v on its boundary. The *right-face* of v is defined analogously. Moreover, we refer to the collection of the left-, right-, and up-faces of v as its *out-faces*, and to the collection of the left-, right-, and down-faces of v as its *in-faces*. The left-to-right ordering of the out-edges (in-edges) of H uniquely determines a left-to-right ordering of the out-faces (in-faces) of H . A hierarchical plane st-graph can be encoded and traversed by means of a slightly augmented DCEL: for each vertex, one simply has to add its y -coordinate and a pointer to its left-most out-edge (except for the sink) and a pointer to its right-most in-edge (except for the source). This augmented DCEL is easily preprocessed in linear time to obtain the peak and valley of each face. Hence, we may assume that each face is equipped with these values. Moreover, we may assume (again via preprocessing in linear time) that the vertices of the outer face are marked, so that it is possible in $\mathcal{O}(1)$ time to test whether a given vertex is external or internal. All hierarchical plane st-graphs in this paper are assumed to be represented by means of this data structure.

We say that an angle is *convex* if it is at most π and *reflex* if it exceeds π . In a *convex* polygon, each internal angle is convex. Recall that a planar straight-line drawing of a graph is called *convex* if the boundary of each face is realized as a simple convex polygon. A *side* of a simple convex polygon is a maximal straight-line segment in its boundary, i.e., a maximal sequence of collinear edges. It is well known that a planar graph admits a convex drawing if and only if it is a subdivision of an *internally 3-connected* graph [31, 30, 20, 19]. There are multiple well-known equivalent definitions of this property. Each of them provides a different perspective on the concept and it will be convenient to refer to all of them. Hence, we define internal 3-connectivity in form of a characterization; a proof of the equivalence of the three properties can be found in [22].

► **Definition 4.** Let G be a plane 2-connected graph and let f_o denote its outer face. Then G is called *internally 3-connected* if and only if the following equivalent statements are satisfied:

- (11) Inserting a new vertex v in f_o and adding edges between v and all vertices of f_o results in a 3-connected graph.
- (12) From each internal vertex w of G there exist three paths to f_o that are pairwise disjoint except for the common vertex w .
- (13) Every separation pair u, v of G is external, meaning that u and v lie on f_o and every connected component of the subgraph of G induced by $V(G) \setminus \{u, v\}$ contains a vertex of f_o .

Let G be a plane graph and Γ° be a convex drawing of the boundary of the outer face of G . Even if G is internally 3-connected, the drawing Γ° cannot necessarily be extended to a convex drawing of G . Recall that a path P of G is *arched* by a face f of G if there exist two distinct vertices $a, b \in V(P) \cap V(\partial f)$ such that the subpath of P between a and b is not a subpath of ∂f , see Figure 2c. Moreover, P is called *archfree* if it is not arched by an internal face of G . The drawing Γ° can be extended to a convex drawing of G if and only if G is a subdivision of an internally 3-connected graph and each side of Γ° corresponds to an archfree path of G [31, 30, 20, 19]. Hong and Nagamochi [20] showed that this characterization carries over to hierarchical graphs: let H be a hierarchical plane st-graph and Λ° be a convex drawing of the restriction of H to the boundary of its outer face. If each side of Λ° corresponds to an archfree path of H , we call Λ° *compatible* with H . The drawing Λ° can be extended to a convex drawing of H if and only if H is a subdivision of an internally 3-connected graph and Λ° is compatible with H [20]. Hong and Nagamochi also proved the following lemma.

► **Lemma 5** ([20, Lemma 1]). *Let G be an internally 3-connected plane graph and let f be an internal face of G . Any subpath P of ∂f with $|E(P)| \leq |E(\partial f)| - 2$ is archfree.*

3 Computing archfree paths efficiently

In this section, we prove Theorem 2, that is, we describe our algorithm for computing archfree paths in internally 3-connected hierarchical plane st-graphs. In fact, we prove the following generalization of Theorem 2, which gives us the freedom to influence the choice of the first edge of the computed path. This aspect will be useful in our algorithm to create convex drawings of hierarchical plane st-graphs.

► **Theorem 6.** *There exists an algorithm that, given an internally 3-connected hierarchical plane st-graph G and (a pointer to) an internal vertex y of G , computes an archfree directed path P_{yz} from y to some outer vertex z of G such that all vertices of P_{yz} except for z are internal vertices of G in time $\mathcal{O}(\sum_{i \in V(P_{yz}) \setminus \{z\}} \deg_G^+(i))$.*

Moreover, the choice of the first edge (y, u) of P_{yz} may be influenced as follows: let F_y be the set of out-faces of y , let $k_y = \max_{g \in F_y} \{\text{peak}(g)\}$, and let $K_y = \{g \in F \mid \text{peak}(g) = k\}$. We may choose (y, u) to be the left-most out-edge of y that is incident to the right-most out-face in K_y ; or the right-most out-edge of y that is incident to the left-most out-face in K_y .

Proof. In the following, we describe the idea of our algorithm; for a full pseudocode version, see Algorithm 1. Suppose we have already computed a directed path P from y to some vertex v such that all vertices of P are internal (initially $v = y$). We will extend P by appending an out-edge e' of v that is incident to an out-face of v with maximum peak, for an illustration see Figure 3. More precisely, let F be the set of out-faces of v , let $k = \max_{g \in F} \{\text{peak}(g)\}$, and, finally, let $K = \{g \in F \mid \text{peak}(g) = k\}$.

(R1) We extend P by appending an out-edge e' of v that is incident to a face $f' \in K$.

At each edge of our path, we store a pointer to the face that was the reason why the edge was chosen, i.e., we *associate* the edge e' with the face f' .

In general, the choice of e' and f' according to Rule (R1) is not unique (see Figure 3), and computing a path while exclusively relying on Rule (R1) does not necessarily lead to an archfree result. We introduce two tiebreaking rules that specialize Rule (R1). Assume for now that $v \neq y$ and let e be the edge of P whose head is v . Let f be the face that is associated with e .

■ **Algorithm 1** The procedure corresponding to Theorem 6.

input : an internally 3-connected hierarchical plane st-graph G
 an internal vertex y of G
 $d \in \{L, R\} \triangleright$ initial associated direction

output : a directed archfree path P_{yz} from y to an outer vertex z of G such that all vertices of P_{yz} except z are internal vertices of G

$v \leftarrow y \triangleright$ The endpoint of our current path P ,
 $P \leftarrow \emptyset \triangleright$ whose list of edges is initially empty.
 $e \leftarrow \text{nil} \triangleright$ The edge that was most recently appended to P
 $f \leftarrow \text{nil} \triangleright$ and the face associated with it.
 $D \leftarrow d \triangleright$ The direction currently associated with P

while true **do**

- \triangleright Let F be the set of out-faces of v
- $k \leftarrow \max_{g \in F} \{\text{peak}(g)\}$
- $K \leftarrow \{g \in F \mid \text{peak}(g) = k\}$
- if** $f \neq \text{nil}$ **and additionally** $\text{peak}(f) = k$ **then**
 - $\mid e \leftarrow$ the unique edge of ∂f with tail v
- else** $\triangleright f = \text{nil}$; or $f \neq \text{nil}$ and additionally $\text{peak}(f) < k$
 - if** $D = L$ **then**
 - $\mid f \leftarrow$ the right-most face in K
 - $\mid e \leftarrow$ the left-most out-edge of v that belongs to ∂f
 - else** $\triangleright D = R$
 - $\mid f \leftarrow$ the left-most face in K
 - $\mid e \leftarrow$ the right-most out-edge of v that belongs to ∂f
- append e to P
- $v \leftarrow \text{head}(e)$
- if** f is to the left of e **then**
 - $\mid D \leftarrow L$
- else** $\triangleright f$ is to the right of e
 - $\mid D \leftarrow R$
- if** v belongs to the outer face of G **then**
 - \mid **return** P

(R2) If $v \neq y$ and $f \in K$, we choose $f' = f$ and e' to be the unique out-edge of v that is incident to f .

In other words, we continue to follow the boundary of a face f until we encounter a vertex v incident to a face with strictly larger peak, see Figure 3a.

It remains to discuss the case where the preconditions of Rule (R2) are not satisfied. We associate a direction $D \in \{L, R\}$ with P , namely, $D = L$ if f is to the left of e , and $D = R$ otherwise. Whenever we switch from f to a new face, we also try to switch the direction associated with our path if possible. That is, if $D = L$, we try to choose f' and e' such that f' is to the right of e' , see Figure 3b. Note that this is impossible if and only if $|K| = 1$ and the unique face $f' \in K$ is the left-face of v , see Figure 3c. Symmetrically, if $D = R$, we try to choose f' and e' such that f' is to the left of e' , which is impossible if and only

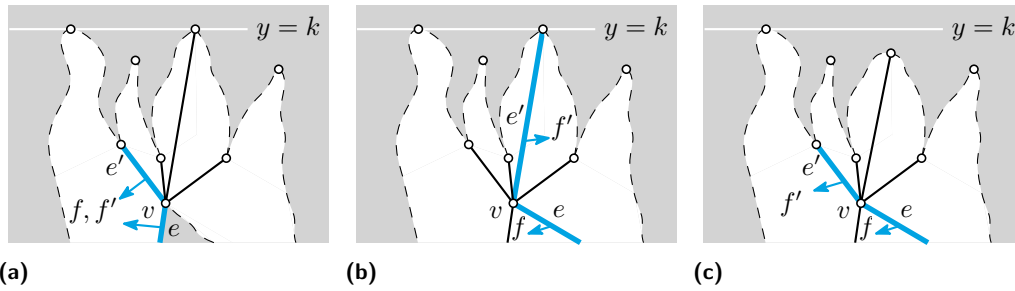


Figure 3 The choices of (e', f') made according to Rule (R1) are sometimes unique (c), but in general they are not (a–b). (a) Unique choice of (e', f') according to Rule (R2). (b–c) Unique choice of (e', f') according to Rule (R3). In (c) it is not possible to switch the direction of the path.

if $|K| = 1$ and the unique $f' \in K$ is the right-face of v . In general, this choice of f' and e' is still not unique. Whenever there are multiple options, we choose f' and e' such that if $D = L$ ($D = R$), the face f' is the right-most (left-most) out-face of v such that the above properties are satisfied, see Figure 3b. To streamline the algorithm, the initial path, where $v = y$ (and, hence, f is undefined), is also associated with a direction L or R . This initial direction may be freely chosen. The following rule realizes the strategy discussed in this paragraph:

(R3) If $v = y$ or $f \notin K$, our choice of f' and e' depends on D : if $D = L$ ($D = R$), we choose f' to be the right-most (left-most) face in K and e' to be the left-most (right-most) out-edge of v that belongs to $\partial f'$.

Indeed, whenever we switch to a new face $f' \neq f$, Rule (R3) ensures that the direction associated with the path is switched if possible:

▷ **Claim 7.** If $D = L$ ($D = R$) and f' and e' are chosen according to Rule (R3), then f' is to the right (left) of e' , unless $|K| = 1$ and the unique $f' \in K$ is the left-face (right-face) of v .

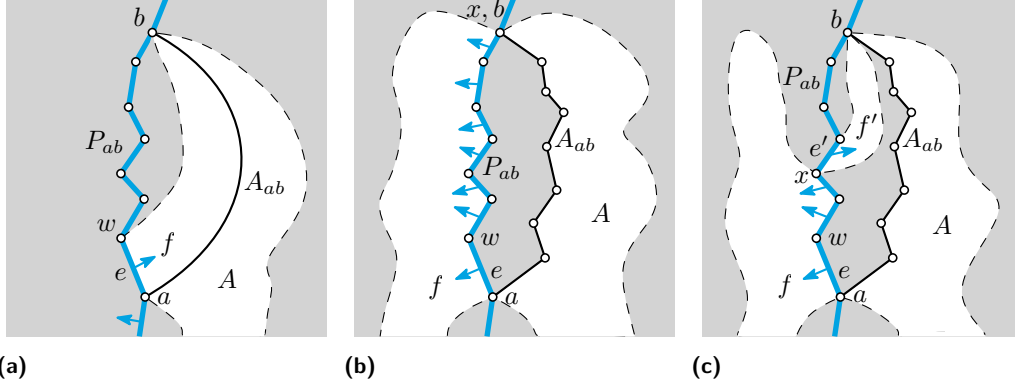
Correctness. Since G has the st-property, Algorithm 1 terminates with a directed path P_{yz} from y to some outer vertex z of G such that all vertices of P except for z are internal vertices of G . Since the initial direction may be freely chosen, Rule (R3) guarantees the choice of the first edge of P_{yz} as described in the statement of the theorem.

It remains to show that P_{yz} is archfree. To the contrary, assume that P_{yz} is arched by a face A . Without loss of generality, we may assume that A is to the right of P_{yz} . This implies the existence of a directed subpath A_{ab} of the left boundary of A that starts at a vertex $a \in V(P_{yz})$, ends at a vertex $b \in V(P_{yz})$ with $b \neq a$, and is interior-disjoint from P_{yz} . Let P_{ab} denote the subpath of P_{yz} that leads from a to b . Let C_{ab} denote the simple cycle formed by P_{ab} and A_{ab} in the underlying undirected graph of G .

In a plane 2-connected graph, every edge belongs to the boundary of at least one internal face. Consequently, Lemma 5 implies that in internally 3-connected graphs every path of length 1 is archfree (since each face is bounded by at least three edges). Since G is internally 3-connected, it follows that $|E(P_{ab})| \geq 2$. Let $e = (a, w)$ denote the unique out-edge of a that belongs to P_{ab} (where $w \neq b$ since $|P_{ab}| \geq 2$). Let f denote the face associated with e . By the definition of A_{ab} , we have $f \neq A$. The face f can be either to the left or the right of e . Accordingly, we distinguish two cases.

Case 1: f is to the right of e . For an illustration see Figure 4a. This implies that f is interior to the cycle C_{ab} . Consequently, we have $\text{peak}(f) \leq \text{peak}(A)$. Moreover, by Rule (R1), we have $\text{peak}(f) = \text{peak}(A)$ since both A and f are out-faces of a . From the fact that f is

interior to the cycle C_{ab} , we can also conclude that a is the (unique) source of f . Therefore, the face f and the edge e were chosen according to Rule (R3) (rather than Rule (R2)). Due to the fact that $\text{peak}(f) = \text{peak}(A)$, Claim 7 implies that the direction associated with the subpath P_{ya} of P_{yz} from y to a is L . Consequently, Rule (R3) implies that f is the right-most out-face of v whose peak is $\text{peak}(f)$ ($= \text{peak}(A)$). We obtain a contradiction to the fact that A is to the right of f . \triangleleft



■ **Figure 4** (a) Case 1 and (b–c) Case 2 in the proof of Theorem 6.

Case 2: f is to the left of e . Let P_{ax} denote the unique maximal subpath of P_{ab} such that all edges of P_{ax} are associated with f .

To the contrary, assume that the endpoint $x \neq a$ of P_{ax} is b ; for an illustration see Figure 4b. It follows that a and b form a separator in G that separates w (and all other vertices in the closed interior of C_{ab} except for a and b) from the outer vertices of G . Consequently, a and b form a nonexternal separation pair; a contradiction to the internal 3-connectivity of G . Therefore, we have $x \neq b$.

Let e' denote the unique out-edge of x that belongs to P_{ab} ; for an illustration see Figure 4c. By Rule (R2), the edge e' is associated with a face f' such that $\text{peak}(f') > \text{peak}(f)$. Therefore, face f' and edge e' were chosen according to Rule (R3). Since f is to the left of e , each edge of P_{ax} has f to its left. Therefore the direction associated with the subpath P_{yx} of P_{yz} from y to x is L . By Rule (R1) applied to a , we have $\text{peak}(f) \geq \text{peak}(A)$. Moreover, we have $\text{peak}(A) > y(x)$ since $x \neq b$. Consequently, $\text{peak}(f) > y(x)$, which implies that f is the left-face of x . Therefore, by Claim 7, the face f' is to the right of e' . This implies that f' is interior to C_{ab} and, consequently, $\text{peak}(f') \leq \text{peak}(A)$. Altogether, we have $\text{peak}(f) < \text{peak}(f') \leq \text{peak}(A) \leq \text{peak}(f)$; a contradiction. \triangleleft

Running time. The claimed running time of $\mathcal{O}(\sum_{i \in V(P_{yz}) \setminus \{z\}} \deg_G^+(i))$ of Algorithm 1 is easy to achieve: the initialization takes $\mathcal{O}(1)$ time. The while loop is executed once for each vertex of P_{yz} that is an internal vertex of G . For each vertex v , the quantity k can be computed in time $\mathcal{O}(\deg_G^+(v))$ by sweeping the linear sequence of out-faces of v once. With a second sweep of the sequence, we can then determine in time $\mathcal{O}(\deg_G^+(v))$ the set K . Actually, it suffices to remember the left-most and the right-most out-face in K . With this information, the remaining lines in the while loop can be executed in time $\mathcal{O}(1)$. \blacktriangleleft

4 Computing convex drawings of hierarchical graphs in linear time

In this section, we describe our $\mathcal{O}(n)$ -time algorithm to create a convex drawing of a hierarchical plane st-graph. We follow the idea of the recursive combinatorial construction by Hong and Nagamochi [20]. As discussed in Section 1, the main improvement comes from using Theorem 6 to compute archfree paths. We also need to make some changes to the low-level details of the original algorithm and apply a more careful (amortized) analysis of its running time. The correctness of our algorithm follows for the most part from the correctness of the algorithm by Hong and Nagamochi.

► **Theorem 1.** *There exists an algorithm that, given a subdivision $G = (V, E)$ of an internally 3-connected hierarchical plane st-graph and a convex polygon Γ° that is compatible with G , computes a convex drawing of G with Γ° as the realization of the outer face in time $\mathcal{O}(n)$ where $n = |V|$.*

Proof sketch. It suffices to prove the claim for the case that G is internally 3-connected. The y -coordinate of each vertex in the desired drawing of G is already fixed. Moreover, the x -coordinates of the vertices of the outer face are also fixed by Γ° . Hence, our goal is to (recursively) compute the x -coordinates of the internal vertices in a convex drawing of G with Γ° as the realization of the outer face.

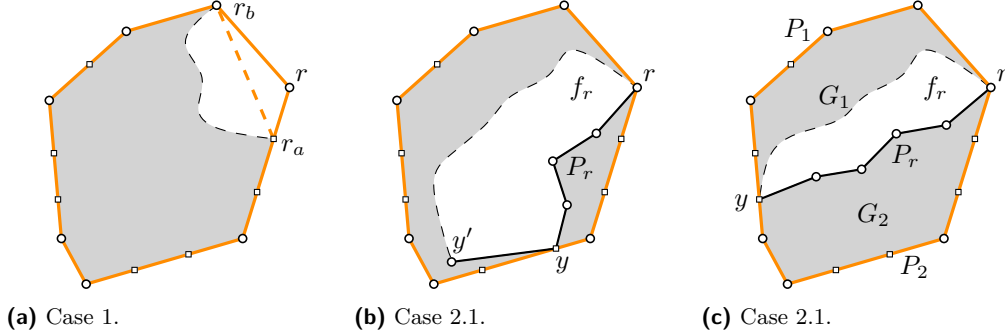
We assume we are given a cyclic list L that contains the vertices of G corresponding to the vertices of Γ° whose outer angles are reflex in the order in which they appear around Γ° . We refer to L as the *corner list* of Γ° . Initially, we can preprocess G and Γ° to obtain L in $\mathcal{O}(n)$ time. When making recursive calls, we will split L appropriately to create a new corner list for each subproblem in $\mathcal{O}(1)$ time. Given the list L , it is possible in $\mathcal{O}(1)$ time to compute a vertex r that corresponds to a vertex of Γ° whose outer angle is reflex and such that r is not the source or sink of G . Without loss of generality, we assume that r belongs to the right boundary of Γ° . We distinguish two cases regarding the degree of r .

Case 1: $\deg_G(r) = 2$. For an illustration see Figure 5a. Let r_a and r_b denote the in-neighbor and out-neighbor of r , respectively. If $|L| = 3$ and r_a is the source of G and r_b is the sink of G , then G has no internal vertices (otherwise, the internal face incident to r would arch the path corresponding to the side $r_a r_b$ of Γ° thereby contradicting the compatibility of Γ°). Hence, we simply report that the coordinates of all internal vertices are already fixed. This is the base case of our recursion. It can be recognized and dealt with in $\mathcal{O}(1)$ time.

So assume that $|L| \geq 4$ or that r has a neighbor that is not the sink or source of G . Let Γ_1° denote the simple (convex) polygon obtained from Γ° created by replacing the segments rr_a and rr_b with the segment $r_a r_b$. The simplicity of Γ_1° is implied by the above assumption. If $(r_a, r_b) \in E$ we set $G'_1 = G$. Otherwise, we add the edge (r_a, r_b) in the internal face incident to r and call the resulting graph G'_1 . We delete r from G'_1 and call the resulting graph G_1 . Properties (I1)–(I2) of Definition 4 can be used to show that G_1 is internally 3-connected. Moreover, it can be derived by Lemma 5 that Γ_1° is compatible with G_1 . We recursively compute the coordinates of the internal vertices in a convex drawing of G_1 with Γ_1° as the realization of the outer face. These coordinates combined with the coordinates of r correspond to the desired drawing of G . The graph G and its polygon Γ° are easily transformed into G_1 and Γ_1° in $\mathcal{O}(1)$ time, and L can be transformed into a corner list of Γ_1° in $\mathcal{O}(1)$ time. ◀

Case 2: $\deg_G(r) \geq 3$. For illustrations see Figures 5b, 5c, and 6a. Without loss of generality, there is an edge whose head is r and whose tail is an internal vertex of G . Let f_r denote the left-face of r and let P'_r denote the (unique) directed subpath of ∂f_r that connects the source y' of f_r with r . Let y be the first outer vertex of G distinct from r that is

encountered when traversing P'_r from r towards y' – if no such vertex exists, we set $y = y'$. We define P_r to be the subpath of P'_r between y and r . If y and r are nonadjacent, we have $|E(P_r)| \leq |E(\partial f_r)| - 2$ and hence P_r is archfree by Lemma 5. If y and r are adjacent, then $P_r = (y, r)$; otherwise, by the definition of f_r , the edge (y, r) is to the right of P_r , which implies that the vertices r, y form a nonexternal separation pair that separates the internal vertices of P_r from the outer face and, thus, contradicts the internal 3-connectivity of G . Hence, P_r is again archfree by Lemma 5. So in any case P_r is archfree. Moreover, P_r is easily computed in $\mathcal{O}(|E(P_r)|)$ time.



■ **Figure 5** Case 1 and Case 2.1 in the proof of Theorem 1 and [20, Theorem 8]. (b) depicts the case where $y \neq y'$ and $P_r \neq P'_r$, whereas (c) depicts the case where $y = y'$ and $P_r = P'_r$.

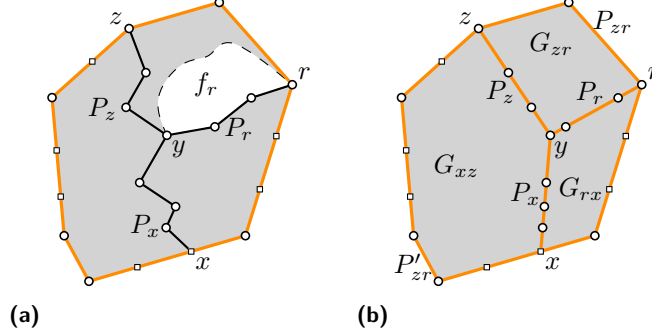
We distinguish two cases depending on whether y is an internal vertex or not.

Case 2.1: y is an outer vertex of G . For illustrations see Figures 5b and 5c. The boundary of the outer face of G contains two interior disjoint paths P_1, P_2 between r and y . Each of these two paths forms a cycle together with P_r . The closed interior of each of these two cycles describes a hierarchical plane st-graph. We denote these two graphs by G_1 and G_2 such that $G_i, i \in \{1, 2\}$ has P_i on its outer face. We define Γ_1^o to be the polygon resulting from replacing the part of Γ^o that corresponds to P_2 with P_r drawn as a straight-line segment, thereby fixing the coordinates of the internal vertices of P_r . The drawing Γ_2^o is defined analogously. Since each side of Γ^o corresponds to an archfree path in G , it follows that r and y do not belong to a common side of Γ^o . Hence, Γ_1^o and Γ_2^o are simple (convex) polygons. Moreover, since P_r is archfree, Γ_1^o is compatible with G_1 and Γ_2^o is compatible with G_2 . By Property (I2) of Definition 4 for G it easily follows that G_1 and G_2 are internally 3-connected. We recursively compute the coordinates of the internal vertices in convex drawings of G_1 and G_2 with outer face Γ_1 and Γ_2 , respectively. Note that these coordinates combined with the coordinates of Γ^o and P_r correspond to the desired drawing of G .

The hierarchical plane st-graphs G_1 and G_2 can be obtained in $\mathcal{O}(|E(P_r)|)$ time by splitting G along P_r . The polygons Γ_1^o and Γ_2^o can be obtained in $\mathcal{O}(|E(P_r)|)$ time by splitting Γ^o and their corner lists can be obtained in $\mathcal{O}(1)$ time by splitting L . \triangleleft

Case 2.2: y is an internal vertex of G . For an illustration see Figure 6a. We compute a directed archfree path P_x from a vertex x on the outer face of G to y by using (a symmetric version of) Theorem 2. The paths P_x and P_r are disjoint except for the common endpoint y . We also compute a directed archfree path P_z from y to a vertex z on the outer face of G by using Theorem 6. To ensure that P_z is disjoint from P_r (except for the common endpoint y), we make use of the ability to influence the choice of the first edge (y, u) of P_z as guaranteed by Theorem 6. Let F_y be the set of out-faces of y , let $k_y = \max_{g \in F_y} \{\text{peak}(g)\}$, and let $K_y = \{g \in F \mid \text{peak}(g) = k\}$. If $K_y = \{f_r\}$, we pick the left-most out-edge of y that is

incident to the right-most out-face in K_y (which is f_r). Since f_r is the left-face of r , this choice guarantees that P_z and P_r are disjoint. Otherwise (if K_y contains at least one face distinct from f_r), we pick the right-most out-edge of y that is incident to the left-most out-face in K_y . Since f_r is the left-face of r , its peak is strictly larger than $y(r)$. Hence, K_y cannot contain any face that is to the right of f_r . Consequently, it contains a face to the left of f_r . Thus, our choice guarantees that P_z is disjoint from P_r since P_r is the left-face of r .



■ **Figure 6** Case 2.2 in the proof of Theorem 1 and [20, Theorem 8].

The boundary of the outer face of G contains two interior disjoint paths from z to r . Let P_{zr} denote the path that does not contain x and let P'_{zr} denote the path that contains x , for an illustration see Figure 6b. The closed interior of the cycle formed by P_{zr} , P_z , and P_r describes a hierarchical plane st-graph G_{zr} , which is easily seen to be internally 3-connected due to Property (I2) of Definition 4. We pick a point p with $y(p) = y(y)$ in the interior of the triangle formed by the vertices r, x, z . Finally, we create a convex polygon Γ^o_{zr} from Γ^o by replacing the part corresponding to P'_{zr} with the straight-line segments pz and pr , which fixes the coordinates of the vertices of P_r and P_z . Since P_r and P_z are archfree in G , it follows that Γ^o_{zr} is compatible with G_{zr} . Analogously, we define two other internally 3-connected hierarchical plane st-graphs G_{rx} and G_{xz} that together with G_{zr} partition the internal faces of G . We also construct two convex polygons Γ^o_{rx} and Γ^o_{xz} that are compatible with G_{rx} and G_{xz} , respectively, that also use the point p as a corner. We recursively compute the internal coordinates of convex drawings of G_{zr} , G_{rx} , and G_{xz} with Γ^o_{zr} , Γ^o_{rx} and Γ^o_{xz} as the realization of the outer face, respectively. Note that these coordinates combined with the coordinates of Γ^o , P_r , P_z , and P_x correspond to the desired drawing of G .

By Theorem 6, the path P_x can be computed in $\mathcal{O}(\sum_{i \in V(P_x) \setminus \{x\}} \deg_G^-(i))$ time and P_z can be computed in $\mathcal{O}(\sum_{i \in V(P_z) \setminus \{z\}} \deg_G^+(i))$ time. By splitting G and Γ^o , we can compute G_{zr} , G_{rx} , and G_{xz} and Γ^o_{zr} , Γ^o_{rx} , and Γ^o_{xz} , respectively, in $\mathcal{O}(|E(P_r)| + |E(P_x)| + |E(P_z)|)$ time. By splitting L , we can compute the corner lists of Γ^o_{zr} , Γ^o_{rx} , and Γ^o_{xz} in $\mathcal{O}(1)$ time. \triangleleft

Running time. The preprocessing (computing L) takes $\mathcal{O}(n)$ time. Case 1 can be dealt with in $\mathcal{O}(1)$ time. The time to take care of Case 2.1 can be expressed as $\mathcal{O}(1) + \mathcal{O}(|E(P_r)|)$. Case 2.2 can be taken care of in time $\mathcal{O}(1) + \mathcal{O}(|E(P_r)| + \sum_{i \in V(P_x) \setminus \{x\}} \deg_G^-(i) + \sum_{i \in V(P_z) \setminus \{z\}} \deg_G^+(i))$. In all three cases, the nonconstant summands only involve internal vertices of G , that is, $|E(P_r)|$ is linear in the number of vertices of P_r that are internal vertices of G , and the expressions $\sum_{i \in V(P_x) \setminus \{x\}} \deg_G^-(i)$ and $\sum_{i \in V(P_z) \setminus \{z\}} \deg_G^+(i)$ only involve the degree of vertices of P_x and P_z , respectively, that are internal to G . The graph G is then split to obtain up to three hierarchical plane st-graphs in which these internal vertices are external. Hence, each vertex of G can contribute to the nonconstant part of the running time of at

most one subproblem in the recursion tree. Therefore, the total sum of these nonconstant parts is bounded by $\sum_{v \in V} \deg_G(v) \subseteq \mathcal{O}(n)$. It remains to find an upper bound on the total number of subproblems. The number of subproblems for which Case 1 arises is bounded by the number of faces of an internal triangulation of G , which is $\mathcal{O}(n)$. Each subproblem for which Case 2 arises splits at least one internal edge, which becomes external in the created subproblems. Hence, each edge is split at most once and so the number of the subproblems for which Case 2 arises is $\mathcal{O}(n)$. Therefore the total running time is $\mathcal{O}(n)$. ◀

References

- 1 Oswin Aichholzer, Greg Aloupis, Erik D. Demaine, Martin L. Demaine, Vida Dujmovic, Ferran Hurtado, Anna Lubiw, Günter Rote, André Schulz, Diane L. Souvaine, and Andrew Winslow. Convexifying polygons without losing visibilities. In Greg Aloupis and Bremner, editors, *Canadian Conference on Computational Geometry (CCCG)*, pages 229–234, 2011.
- 2 Soroush Alamdari, Patrizio Angelini, Fidel Barrera-Cruz, Timothy M. Chan, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Penny Haxell, Anna Lubiw, Maurizio Patrignani, Vincenzo Roselli, Sahil Singla, and Bryan T. Wilkinson. How to morph planar graph drawings. *SIAM J. Comput.*, 46(2):824–852, 2017. doi:10.1137/16M1069171.
- 3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
- 4 Noga Alon and Raphael Yuster. Matrix sparsification and nested dissection over arbitrary fields. *Journal of the ACM*, 60(4):25, 2013.
- 5 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. doi:10.1016/j.tcs.2014.12.019.
- 6 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli. Morphing planar graph drawings optimally. 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 126–137. Springer, 2014. doi:10.1007/978-3-662-43948-7_11.
- 7 Patrizio Angelini, Giordano Da Lozzo, Fabrizio Frati, Anna Lubiw, Maurizio Patrignani, and Vincenzo Roselli. Optimal morphs of convex drawings. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 126–140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.SOCG.2015.126.
- 8 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. 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 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.
- 9 Erin Wolf Chambers, Jeff Erickson, Patrick Lin, and Salman Parsa. How to morph graphs on the torus. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2759–2778. SIAM, 2021. doi:10.1137/1.9781611976465.164.
- 10 Marek Chrobak, Michael T. Goodrich, and Roberto Tamassia. Convex drawings of graphs in two and three dimensions (preliminary version). In Sue Whitesides, editor, *Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, May 24-26, 1996*, pages 319–328. ACM, 1996. doi:10.1145/237218.237401.
- 11 Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30:205–239, 2003.

- 12 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 13 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.
- 14 Peter Eades, Qing-Wen Feng, Xuemin Lin, and Hiroshi Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006. doi:10.1007/s00453-004-1144-8.
- 15 Michael S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.
- 16 Michael S. Floater. Parametric tilings and scattered data approximation. *International Journal of Shape Modeling*, 4(03n04):165–182, 1998.
- 17 Jonas Gomes, Lucia Darsa, Bruno Costa, and Luiz Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.
- 18 Lenwood S. Heath and Sriram V. Pemmaraju. Recognizing leveled-planar dags in linear time. In Franz-Josef Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 1995. doi:10.1007/BFb0021813.
- 19 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of graphs with non-convex boundary constraints. *Discret. Appl. Math.*, 156(12):2368–2380, 2008. doi:10.1016/j.dam.2007.10.012.
- 20 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *J. Discrete Algorithms*, 8(3):282–295, 2010. doi:10.1016/j.jda.2009.05.003.
- 21 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level planarity testing in linear time. In Sue Whitesides, editor, *Graph Drawing, 6th International Symposium, GD'98, Montréal, Canada, August 1998, Proceedings*, volume 1547 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 1998. doi:10.1007/3-540-37623-2_17.
- 22 Linda Kleist, Boris Klemz, Anna Lubiw, Lena Schlipf, Frank Staals, and Darren Strash. Convexity-increasing morphs of planar graphs. *Comput. Geom.*, 84:69–88, 2019. doi:10.1016/j.comgeo.2019.07.007.
- 23 Boris Klemz and Günter Rote. Ordered level planarity and its relationship to geodesic planarity, bi-monotonicity, and variations of level planarity. *ACM Trans. Algorithms*, 15(4):53:1–53:25, 2019. doi:10.1145/3359587.
- 24 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, 2014.
- 25 Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM J. Numerical Analysis*, 16(2):346–358, 1979.
- 26 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046.
- 27 Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli. Upward planar morphs. *Algorithmica*, 82(10):2985–3017, 2020. doi:10.1007/s00453-020-00714-6.
- 28 János Pach and Géza Tóth. Monotone drawings of planar graphs. *J. Graph Theory*, 46(1):39–47, 2004. doi:10.1002/jgt.10168.
- 29 André Schulz. Drawing 3-polytopes with good vertex resolution. *J. Graph Algorithms Appl.*, 15(1):33–52, 2011. doi:10.7155/jgaa.00216.
- 30 Carsten Thomassen. Plane representations of graphs. In J. A. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory*, pages 43–69. Academic Press, 1984.
- 31 William T. Tutte. Convex representations of graphs. *Proceedings of the London Mathematical Society*, s3-10(1):304–320, 1960. doi:10.1112/plms/s3-10.1.304.
- 32 William T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.

QCSP on Reflexive Tournaments

Benoît Larose ✉

LACIM, University of Québec, Montréal, Canada

Petar Marković ✉

Department of Mathematics and Informatics, University of Novi Sad, Serbia

Barnaby Martin ✉

Department of Computer Science, Durham University, UK

Daniël Paulusma ✉

Department of Computer Science, Durham University, UK

Siani Smith ✉

Department of Computer Science, Durham University, UK

Stanislav Živný ✉

Department of Computer Science, University of Oxford, UK

Abstract

We give a complexity dichotomy for the Quantified Constraint Satisfaction Problem QCSP(H) when H is a reflexive tournament. It is well-known that reflexive tournaments can be split into a sequence of strongly connected components H_1, \dots, H_n so that there exists an edge from every vertex of H_i to every vertex of H_j if and only if $i < j$. We prove that if H has both its initial and final strongly connected component (possibly equal) of size 1, then QCSP(H) is in NL and otherwise QCSP(H) is NP-hard.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases computational complexity, algorithmic graph theory, quantified constraints, universal algebra, constraint satisfaction

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.58

Related Version *Full Version:* <https://arxiv.org/abs/2104.10570>

Funding *Stanislav Živný:* Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors’ views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

Acknowledgements We are grateful to several referees for careful reading of the paper and good advice.

1 Introduction

The *Quantified Constraint Satisfaction Problem* QCSP(B), for a fixed *template* (structure) B, is a popular generalisation of the *Constraint Satisfaction Problem* CSP(B). In the latter, one asks if a primitive positive sentence (the existential quantification of a conjunction of atoms) φ is true on B, while in the former this sentence may also have universal quantification. Much of the theoretical research into (finite-domain¹) CSPs has been in respect of a complexity classification project [11, 5], recently completed by [4, 22, 24], in which it is shown that all

¹ All structures considered in this article are finite.



such problems are either in P or NP-complete. Various methods, including combinatorial (graph-theoretic), logical and universal-algebraic were brought to bear on this classification project, with many remarkable consequences.

Complexity classifications for QCSPs appear to be harder than for CSPs. Indeed, a classification for QCSPs will give a fortiori a classification for CSPs (if $B \uplus K_1$ is the disjoint union of B with an isolated element, then $\text{QCSP}(B \uplus K_1)$ and $\text{CSP}(B)$ are polynomial-time many-one equivalent). Just as $\text{CSP}(B)$ is always in NP, so $\text{QCSP}(B)$ is always in Pspace. However, no polychotomy has been conjectured for the complexities of $\text{QCSP}(B)$, though, until recently, only the complexities P, NP-complete and Pspace-complete were known. Recent work [25] has shown that this complexity landscape is considerably richer, and that dichotomies of the form P versus NP-hard (using Turing reductions) might be the sensible place to be looking for classifications.

$\text{CSP}(B)$ may equivalently be seen as the *homomorphism* problem which takes as input a structure A and asks if there is a homomorphism from A to B . The *surjective CSP*, $\text{SCSP}(B)$, is a cousin of $\text{CSP}(B)$ in which one requires that this homomorphism from A to B be surjective. From the logical perspective this translates to the stipulation that all elements of B be used as witnesses to the (existential) variables of the primitive positive input φ . The surjective CSP appears in the literature under a variety of names, including *surjective homomorphism* [2], *surjective colouring* [12, 15] and *vertex compaction* [19, 20]. $\text{CSP}(B)$ and $\text{SCSP}(B)$ have various other cousins: see the survey [2] or, in the specific context of reflexive tournaments, [15]. The only one we will dwell on here is the *retraction* problem $\text{CSP}^c(B)$ which can be defined in various ways but, in keeping with the present narrative, we could define logically as allowing atoms of the form $v = b$ in the input sentence φ where b is some element of B (the superscript c indicates that constants are allowed). It has only recently been shown that there exists a B so that $\text{SCSP}(B)$ is in P while $\text{CSP}^c(B)$ is NP-complete [23]. It is still not known whether such an example exists among the (partially reflexive) graphs.

It is well-known that the binary *cousin* relation is not transitive, so let us ask the question as to whether the surjective CSP and QCSP are themselves cousins? The algebraic operations pertaining to the CSP are *polymorphisms* and for QCSP these become *surjective polymorphisms*. On the other hand, a natural use of universal quantification in the QCSP might be to ensure some kind of surjective map (at least under some evaluation of many universally quantified variables). So it is that there may appear to be some relationship between the problems. Yet, there are known irreflexive graphs H for which $\text{QCSP}(H)$ is in NL, while $\text{SCSP}(H)$ is NP-complete (take the 6-cycle [18, 20]). On the other hand, one can find a 3-element B whose relations are preserved by a *semilattice-without-unit* operation such that both $\text{CSP}^c(B)$ and $\text{SCSP}(B)$ are in P but $\text{QCSP}(B)$ is Pspace-complete. We are not aware of examples like this among graphs and it is perfectly possible that for (partially reflexive) graphs H , $\text{SCSP}(H)$ being in P implies that $\text{QCSP}(H)$ is in P.

Tournaments, both irreflexive and reflexive (and sometimes in between), have played a strong role as a testbed for conjectures and a habitat for classifications, for relatives of the CSP both complexity-theoretic [1, 10, 15] and algebraic [14, 21]. Looking at Table 1 one can see the last unresolved case is precisely QCSP on reflexive tournaments. This is the case we address in this paper. For irreflexive tournaments H , $\text{QCSP}(H)$ is in P if and only if $\text{SCSP}(H)$ is in P, but for reflexive tournaments this is not the case. When H is a reflexive tournament, we prove that $\text{QCSP}(H)$ is in NL if H has both initial and final strongly connected components trivial, and is NP-hard otherwise. In contrast to the proof from [10] and like the proof of [15], we will henceforth work largely combinatorially rather than algebraically. Note that we

do not investigate beyond NP-hard, so our dichotomy cannot be compared directly to the trichotomy of [10] for irreflexive tournaments which distinguishes between P, NP-complete and Pspace-complete.

■ **Table 1** Our result in a wider context. The results for irreflexive tournaments were all proved in the more general setting of irreflexive semicomplete digraphs in the papers cited.

	QCSP	CSP	Surjective CSP	Retraction
irreflexive tournaments	trichotomy [10]	dichotomy [1]	dichotomy [1]	dichotomy [1]
reflexive tournaments	this paper	all trivial	dichotomy [15]	dichotomy [14]

In Section 3 we prove the NP-hard cases of our dichotomy. Our proof method follows that from [15], while adapting the ideas of [8] in order to make what was developed for Surjective CSP applicable to QCSP. The QCSP is not naturally a combinatorial problem but can be seen thusly when viewed in a certain way. We indeed closely mirror [15] with [8] in the strongly connected case. For the not strongly connected case, the adaptation from the strongly connected case was straightforward for the Surjective CSP in [15]. However, the straightforward method does not work for the QCSP. Instead, we seek a direct argument that essentially sees us extending the method from [15].

In Section 4 we prove the NL cases of our dichotomy. Here, we use ideas originally developed in (the conference version of) [8] and first used in the wild in [17]. Thus, we do not introduce new proof techniques as such but rather weave our proof through the reasonably intricate synthesis of different known techniques. In Section 5 we state our dichotomy and give some directions for future work. Owing to space restrictions in the original submission, some of our proofs are omitted.

2 Preliminaries

For an integer $k \geq 1$, we write $[k] := \{1, \dots, k\}$. A vertex $u \in V(G)$ in a digraph G is *backwards-adjacent* to another vertex $v \in V$ if $(u, v) \in E$. It is *forwards-adjacent* to another vertex $v \in V$ if $(v, u) \in E$. If a vertex u has a self-loop (u, u) , then u is *reflexive*; otherwise u is *irreflexive*. A digraph G is *reflexive* or *irreflexive* if all its vertices are reflexive or irreflexive, respectively.

The *directed path* on k vertices is the digraph with vertices u_0, \dots, u_{k-1} and edges (u_i, u_{i+1}) for $i = 0, \dots, k-2$. By adding the edge (u_{k-1}, u_0) , we obtain the *directed cycle* on k vertices. A digraph G is *strongly connected* if for all $u, v \in V(G)$ there is a directed path in $E(G)$ from u to v . A *double edge* in a digraph G consists in a pair of distinct vertices $u, v \in V(G)$, so that (u, v) and (v, u) belong to $E(G)$. A digraph G is *semicomplete* if for every two distinct vertices u and v , at least one of (u, v) , (v, u) belongs to $E(G)$. A semicomplete digraph G is a *tournament* if for every two distinct vertices u and v , exactly one of (u, v) , (v, u) belongs to $E(G)$. A reflexive tournament G is *transitive* if for every three vertices u, v, w with $(u, v), (v, w) \in E(G)$, also (u, w) belongs to $E(G)$. A digraph F is a *subgraph* of a digraph G if $V(F) \subseteq V(G)$ and $E(F) \subseteq E(G)$. It is *induced* if $E(F)$ coincides with $E(G)$ restricted to pairs containing only vertices of $V(F)$. A *subtournament* is an induced subgraph of a tournament. It is well known that a reflexive tournament H can be split into a sequence of strongly connected components H_1, \dots, H_n for some integer $n \geq 1$ so that there exists an edge from every vertex of H_i to every vertex of H_j if and only if $i < j$. We will use the notation $H_1 \Rightarrow \dots \Rightarrow H_n$ for H and we refer to H_1 and H_n as the *initial* and *final* components of H , respectively.

A *homomorphism* from a digraph G to a digraph H is a function $f : V(G) \rightarrow V(H)$ such that for all $u, v \in V(G)$ with $(u, v) \in E(G)$ we have $(f(u), f(v)) \in E(H)$. We say that f is *(vertex)-surjective* if for every vertex $x \in V(H)$ there exists a vertex $u \in V(G)$ with $f(u) = x$. A digraph H' is a *homomorphic image* of a digraph H if there is a surjective homomorphism from H to H' that is also *edge-surjective*, that is, for all $(x', y') \in E(H')$ there exists an $(x, y) \in E(H)$ with $x' = h(x)$ and $y' = h(y)$.

The problem H-RETRACTION takes as input a graph G of which H is an induced subgraph and asks whether there is a homomorphism from G to H that is the identity on H . This definition is polynomial-time many-one equivalent to the one we suggested in the introduction (see e.g. [2]). The *quantified constraint satisfaction problem* QCSP(H) takes as input a sentence $\varphi := \forall x_1 \exists y_1 \dots \forall x_n \exists y_n \Phi(x_1, y_1, \dots, x_n, y_n)$, where Φ is a conjunction of positive atomic (binary edge) relations. This is a yes-instance to the problem just in case $H \models \varphi$.

The *canonical query* of G (from [13]) is a primitive positive sentence φ_G that has the property that, for all H , G has a homomorphism to H iff $H \models \varphi_G$. It is built by mapping edges (x, y) from $E(G)$ to atoms $E(x, y)$ is an existentially quantified conjunctive query.

The *direct product* of two digraphs G and H , denoted $G \times H$, is the digraph on vertex set $V(G) \times V(H)$ with edges $((x, y), (x', y'))$ if and only if $(x, x') \in E(G)$ and $(y, y') \in E(H)$. We denote the direct product of k copies of G by G^k . A *k-ary polymorphism* of G is a homomorphism f from G^k to G ; if $k = 1$, then f is also called an *endomorphism*. A *k-ary polymorphism* f is *essentially unary* if there exists a unary operation g and $i \in [k]$ so that $f(x_1, \dots, x_k) = g(x_i)$ for every $(x_1, \dots, x_k) \in G^k$. Let us say that a *k-ary polymorphism* f is *uniformly* z for some $z \in V(G)$ if $f(x_1, \dots, x_k) = z$ for every $(x_1, \dots, x_k) \in V(G^k)$. We need the following two lemmas.

► **Lemma 1.** *Let H be a reflexive tournament and f be a k -ary polymorphism of H . If $f(x, \dots, x) = z$ for every $x \in V(H)$, then f is uniformly z .*

Proof. Consider some tuple (x_1, \dots, x_k) which has m distinct vertices. We proceed by induction on m , where the base case $m = 1$ is given as an assumption. Suppose we have the result for m vertices and let (x_1, \dots, x_k) have $m + 1$ distinct entries. For simplicity (and w.l.o.g.) we will consider this reordered and without duplicates as $(y_1, \dots, y_m, y_{m+1})$. Suppose f maps (x_1, \dots, x_k) to z' . Assume $(y_m, y_{m+1}) \in E(H)$ (the case (y_{m+1}, y_m) is symmetric). Then consider the tuples (y_1, \dots, y_m, y_m) and $(y_1, \dots, y_{m+1}, y_{m+1})$. By the inductive hypothesis, f maps each of these (when reordered and padded appropriately with duplicates) to z . Furthermore, we have co-ordinatewise edges from (y_1, \dots, y_m, y_m) to $(y_1, \dots, y_m, y_{m+1})$ and from $(y_1, \dots, y_m, y_{m+1})$ to $(y_1, \dots, y_{m+1}, y_{m+1})$. Since we deduce by the definition of polymorphism that both $(z, z'), (z', z) \in E(H)$, it follows that $z' = z$. Thus, f maps also $(y_1, \dots, y_m, y_{m+1})$ (when reordered and padded appropriately with duplicates) to z . That is, $f(x_1, \dots, x_k) = z$. ◀

► **Lemma 2.** *Let H be the reflexive tournament $H_1 \Rightarrow \dots \Rightarrow H_i \Rightarrow \dots \Rightarrow H_n$. If f is a k -ary surjective polymorphism of H , then f preserves each of $V(H_1), \dots, V(H_n)$; that is, for every i and every tuple of k vertices $x_1, \dots, x_k \in V(H_i)$, $f(x_1, \dots, x_k) \in V(H_i)$.*

Proof. Suppose f maps some tuple (x_1, \dots, x_m) from $V(H_i)$ to $y \in V(H_\ell)$. Let (x'_1, \dots, x'_m) be any tuple from $V(H_i)$. Since H_i is strongly connected, $f(x'_1, \dots, x'_m) \in V(H_\ell)$. It follows that if $\ell \neq i$, e.g. w.l.o.g. $\ell < i$, then some component $\ell' \geq i$ can not be in the range of f . ◀

The relevance of this lemma is in its sequent corollary, which follows according to Proposition 3.15 of [3].

► **Corollary 3.** *Let H be the reflexive tournament $H_1 \Rightarrow \dots \Rightarrow H_i \Rightarrow \dots \Rightarrow H_n$. Each subset of the domain $V(H_i)$ is definable by a QCSP instance in one free variable.*

An endomorphism e of a digraph G is a *constant map* if there exists a vertex $v \in V(G)$ such that $e(u) = v$ for every $u \in V(G)$, and e is the *identity* if $e(u) = u$ for every $u \in G$. An *automorphism* is a bijective endomorphism whose inverse is a homomorphism. An endomorphism is *trivial* if it is either an automorphism or a constant map; otherwise it is *non-trivial*. A digraph is *endo-trivial* if all of its endomorphisms are trivial. An endomorphism e of a digraph G *fixes* a subset $S \subseteq V(G)$ if $e(S) = S$, that is, $e(x) \in S$ for every $x \in S$, and e *fixes* an induced subgraph F of G if it is the identity on $V(F)$. It *fixes* an induced subgraph F *up to automorphism* if $e(F)$ is an automorphic copy of F . An endomorphism e of G is a *retraction* of G if e is the identity on $e(V(G))$. A digraph is *retract-trivial* if all of its retractions are the identity or constant maps. Note that endo-triviality implies retract-triviality, but the reverse implication is not necessarily true (see [15]). However, on reflexive tournaments both concepts do coincide [15].

We need a series of results from [15]. The third one follows from the well-known fact that every strongly connected tournament has a directed Hamilton cycle [6].

► **Lemma 4** ([15]). *A reflexive tournament is endo-trivial if and only if it is retract-trivial.*

► **Lemma 5** ([15]). *Let H be an endo-trivial reflexive digraph with at least three vertices. Then every polymorphism of H is essentially unary.*

► **Lemma 6** ([15]). *If H is an endo-trivial reflexive tournament, then H contains a directed Hamilton cycle.*

► **Lemma 7** ([15]). *If H is an endo-trivial reflexive tournament, then every homomorphic image of H of size $1 < n < |V(H)|$ has a double edge.*

► **Corollary 8.** *If H is an endo-trivial reflexive digraph on at least three vertices, then $\text{QCSP}(H)$ is NP-hard (in fact it is even Pspace-complete).*

Proof. This follows from Lemma 5 and [3]. ◀

3 The Proof of the NP-Hard Cases of the Dichotomy

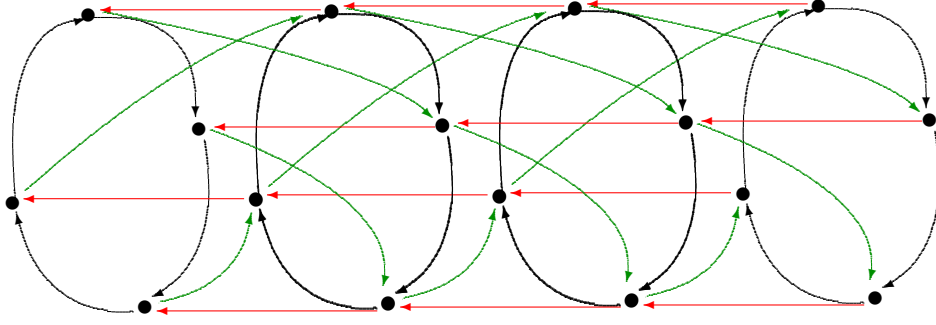
We commence with the NP-hard cases of the dichotomy. The simpler NL cases will follow.

3.1 The NP-Hardness Gadget

We introduce the gadget Cyl_m^* from [15] drawn in Figure 1. Take m disjoint copies of the (reflexive) directed m -cycle DC_m^* arranged in a cylindrical fashion so that there is an edge from i in the j th copy to i in the $(j+1)$ th copy (drawn in red), and an edge from i in the $(j+1)$ th copy to $(i+1) \bmod m$ in the j th copy (drawn in green). We consider DC_m^* to have vertices $\{1, \dots, m\}$. Recall that every strongly connected (reflexive) tournament on m vertices has a Hamilton Cycle HC_m . We label the vertices of HC_m as $1, \dots, m$ in order to attach it to the gadget Cyl_m^* .²

The following lemma follows from induction on the copies of DC_m^* , since a reflexive tournament has no double edges.

² The superscripted $*$ indicates that the corresponding graph is reflexive. This notation is inherited from [15]. It is not significant since we could safely assume every graph we work with is reflexive as the template is a reflexive tournament.



■ **Figure 1** The gadget Cyl_m^* in the case $m := 4$ (self-loops are not drawn). We usually visualise the right-hand copy of DC_4^* as the “bottom” copy and then we talk about vertices “above” and “below” according to the red arrows.

► **Lemma 9** ([15]). *In any homomorphism h from Cyl_m^* , with bottom cycle DC_m^* , to a reflexive tournament, if $|h(\text{DC}_m^*)| = 1$, then $|h(\text{Cyl}_m^*)| = 1$.*

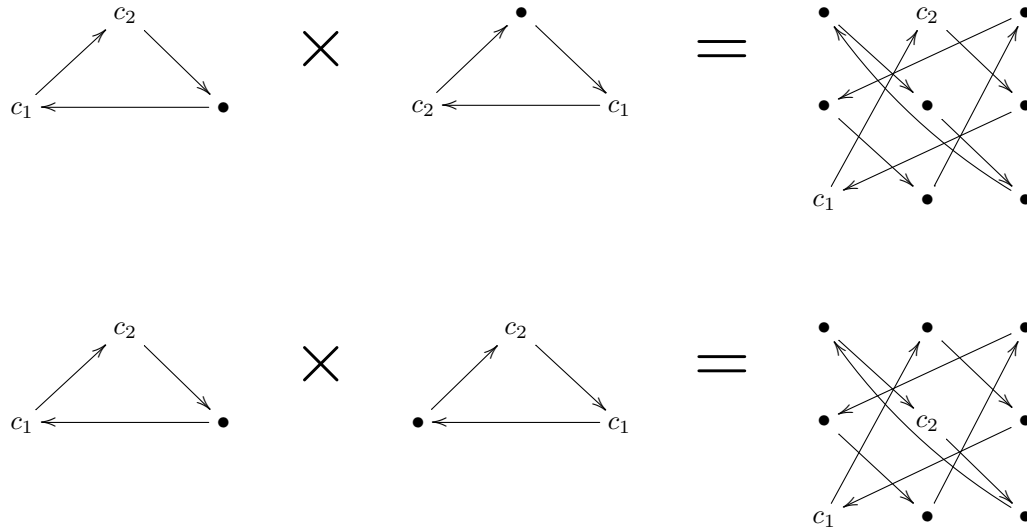
We will use another property, denoted (\dagger) , of Cyl_m^* , which is that the retractions from Cyl_m^* to its bottom copy of DC_m^* , once propagated through the intermediate copies, induce on the top copy precisely the set of automorphisms of DC_m^* . That is, the top copy of DC_m^* is mapped isomorphically to the bottom copy, and all such isomorphisms may be realised. The reason is that in such a retraction, the $(j+1)$ th copy may either map under the identity to the j th copy, or rotate one edge of the cycle clockwise, and Cyl_m^* consists of sufficiently many (namely m) copies of DC_m^* . Now let H be a reflexive tournament that contains a subtournament H_0 on m vertices that is endo-trivial. By Lemma 6, we find that H_0 contains at least one directed Hamilton cycle HC_0 . Define $\text{Spill}_m(H[H_0, \text{HC}_0])$ as follows. Begin with H and add a copy of the gadget Cyl_m^* , where the bottom copy of DC_m^* is identified with HC_0 , to build a digraph $F(H_0, \text{HC}_0)$. Now ask, for some $y \in V(H)$ whether there is a retraction r of $F(H_0, \text{HC}_0)$ to H so that some vertex x (not dependent on y) in the top copy of DC_m^* in Cyl_m^* is such that $r(x) = y$. Such vertices y comprise the set $\text{Spill}_m(H[H_0, \text{HC}_0])$.

► **Remark 10.** If x belongs to some copy of DC_m^* that is not the top copy, we can find a vertex x' in the top copy of DC_m^* and a retraction r' from $F(H_0, \text{HC}_0)$ to H with $r'(x') = r(x) = y$, namely by letting r' map the vertices of higher copies of DC_m^* to the image of their corresponding vertex in the copy that contains x . In particular this implies that $\text{Spill}_m(H[H_0, \text{HC}_0])$ contains $V(H_0)$.

We note that the set $\text{Spill}_m(H[H_0, \text{HC}_0])$ is potentially dependent on which Hamilton cycle in H_0 is chosen. We now recall that $\text{Spill}_m(H[H_0, \text{HC}_0]) = V(H)$ if H retracts to H_0 .

► **Lemma 11** ([15]). *If H is a reflexive tournament that retracts to a subtournament H_0 with Hamilton cycle HC_0 , then $\text{Spill}_m(H[H_0, \text{HC}_0]) = V(H)$.*

We now review a variant of a construction from [8]. Let G be a graph containing H where $|V(H)|$ is of size n . Consider all possible functions $\lambda : [n] \rightarrow V(H)$ (let us write $\lambda \in V(H)^{[n]}$ of cardinality N). For some such λ , let $\mathcal{G}(\lambda)$ be the graph G enriched with constants c_1, \dots, c_n where these are interpreted over $V(H)$ according to λ in the natural way (acting on the subscripts). We use calligraphic notation to remind the reader the signature has changed from $\{E\}$ to $\{E, c_1, \dots, c_n\}$ but we will still treat these structures as graphs. If we write $G(\lambda)$ without calligraphic notation we mean we look at only the $\{E\}$ -reduct, that is, we drop the constants. Of course, $G(\lambda)$ will always be G .



■ **Figure 2** Illustrations of direct product with constants.

Let $\mathcal{G} = \bigotimes_{\lambda \in V(H)^{[n]}} \mathcal{G}(\lambda)$. That is, the vertices of \mathcal{G} are N -tuples over $V(G)$ and there is an edge between two such vertices (x_1, \dots, x_N) and (y_1, \dots, y_N) if and only if $(x_1, y_1), \dots, (x_N, y_N) \in E(G)$. Finally, the constants c_i are interpreted as (x_1, \dots, x_N) so that $\lambda_1(c_i) = x_1, \dots, \lambda_N(c_i) = x_N$. An important induced substructure of \mathcal{G} is $\{(x, \dots, x) : x \in V(G)\}$. It is a copy of G called the *diagonal copy* and will play an important role in the sequel. To comprehend better the construction of \mathcal{G} from the sundry $\mathcal{G}(\lambda)$, confer on Figure 2.

The final ingredient of our fundamental construction involves taking some structure \mathcal{G} and making its canonical query with all vertices other than those corresponding to c_1, \dots, c_n becoming existentially quantified variables (as usual in this construction). We then turn the c_1, \dots, c_n to variables y_1, \dots, y_n to make $\varphi_{\mathcal{G}}(y_1, \dots, y_n)$. Let \mathcal{H} come from the given construction in which $G = H$. It is proved in [8] that $H' \models \forall y_1, \dots, y_n \varphi_{\mathcal{H}}(y_1, \dots, y_n)$ if and only if $\text{QCSP}(H) \subseteq \text{QCSP}(H')$ (here we identify $\text{QCSP}(H)$ with the set of sentences that form its yes-instances). By way of a side note, let us consider a k -ary relation R over H with tuples $(x_1^1, \dots, x_k^1), \dots, (x_1^r, \dots, x_k^r)$. For $i \in [r]$, let λ_i map (c_1, \dots, c_k) to (x_1^i, \dots, x_k^i) . Let $\mathcal{H} = \bigotimes_{\lambda \in \{\lambda_1, \dots, \lambda_r\}} \mathcal{H}(\lambda)$. Then $\varphi_{\mathcal{H}}(y_1, \dots, y_n)$ is the closure of R under the polymorphisms of H .

3.2 The strongly connected case: Two Base Cases

Recall that if H is a (reflexive) endo-trivial tournament, then $\text{QCSP}(H)$ is NP-hard due to Lemma 5 combined with the results from [3] (indeed, we may even say Pspace-complete). However H may not be endo-trivial. We will now show how to deal with the case where H is not endo-trivial but retracts to an endo-trivial subtournament. For doing this we use the NP-hardness gadget, but we need to distinguish between two different cases.

► **Lemma 12** (Base Case I.). *Let H be a reflexive tournament that retracts to an endo-trivial subtournament H_0 with Hamilton cycle HC_0 . Assume that H retracts to H'_0 for every isomorphic copy $H'_0 = i(H_0)$ of H_0 in H with $\text{Spill}_m(H[H'_0, i(\text{HC}_0)]) = V(H)$. Then H_0 -RETRACTION can be polynomially reduced to $\text{QCSP}(H)$.*

Proof. Let m be the size of $|V(H_0)|$ and n be the size of $|V(H)|$. Let G be an instance of H_0 -RETRACTION. We build an instance φ of $\text{QCSP}(H)$ in the following fashion. First, take a copy of H together with G and build G' by identifying these on the copy of H_0 that they both possess as an induced subgraph. Now, consider all possible functions $\lambda : [n] \rightarrow V(H)$. For some such λ , let $\mathcal{G}'(\lambda)$ be the graph enriched with constants c_1, \dots, c_n where these are interpreted over some subset of $V(H)$ according to λ in the natural way (acting on the subscripts).

Let $\mathcal{G}' = \bigotimes_{\lambda \in V(H)^{[n]}} \mathcal{G}'(\lambda)$. Let G'^d , H^d and H_0^d be the diagonal copies of G' , H and H_0 in \mathcal{G}' . Let \mathcal{H} be the subgraph of \mathcal{G}' induced by $V(H) \times \dots \times V(H)$. Note that the constants c_1, \dots, c_n live in \mathcal{H} . Now build \mathcal{G}'' from \mathcal{G}' by augmenting a new copy of Cyl_m^* for every vertex $v \in V(\mathcal{H}) \setminus V(H_0^d)$. Vertex v is to be identified with any vertex in the top copy of DC_m^* in Cyl_m^* and the bottom copy of DC_m^* is to be identified with HC_0 in H_0^d according to the identity function. (Thus, in each case, the new vertices are the middle cycles of Cyl_m^* and all but one of the vertices in the top cycle of Cyl_m^* .)

Finally, build φ from the canonical query of \mathcal{G}'' where we additionally turn the constants c_1, \dots, c_n to outermost universal variables. The size of φ is doubly exponential in n (the size of H) but this is constant, so still polynomial in the size of G .

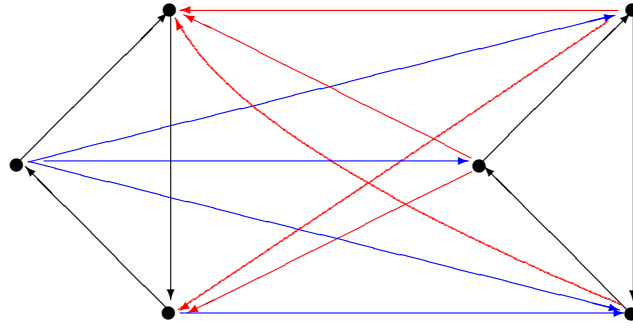
We claim that G retracts to H_0 if and only if $\varphi \in \text{QCSP}(H)$.

First suppose that G retracts to H_0 . Let λ be some assignment of the universal variables of φ to H . To prove $\varphi \in \text{QCSP}(H)$ it suffices to prove that there is a homomorphism from \mathcal{G}'' to H that extends λ . Then for this it suffices to prove that there is a homomorphism h from \mathcal{G}' that extends λ . Let us explain why. Because H retracts to H_0 , we have $\text{Spill}_m(H[H_0, \text{HC}_0]) = V(H)$ due to Lemma 11. Hence, if $h(x) = y$ for two vertices $x \in V(\mathcal{H}) \setminus V(H_0^d)$ and $y \in V(H)$, we can always find a retraction of the graph $F(H_0, \text{HC}_0)$ to H that maps x to y , and we mimic this retraction on the corresponding subgraph in \mathcal{G}'' . The crucial observation is that this can be done independently for each vertex in $V(\mathcal{H}) \setminus V(H_0^d)$, as two vertices of different copies of Cyl_m^* are only adjacent if they both belong to \mathcal{H} .

Henceforth let us consider the homomorphic image of \mathcal{G}' that is $\mathcal{G}'(\lambda)$. To prove $\varphi \in \text{QCSP}(H)$ it suffices to prove that there is a homomorphism from $G'(\lambda)$ to H that extends λ . Note that it will be sufficient to prove that G' retracts to H . Let h be the natural retraction from G' to H that extends the known retraction from G to H_0 . We are done.

Suppose now $\varphi \in \text{QCSP}(H)$. Choose some surjection for λ , the assignment of the universal variables of φ to H . Recall $N = |V(H)^{[n]}|$. The evaluation of the existential variables that witness $\varphi \in \text{QCSP}(H)$ induces a surjective homomorphism s from \mathcal{G}'' to H which contains within it a surjective homomorphism s' from $\mathcal{H} = H^N$ to H . Consider the diagonal copy of $H_0^d \subset H^d \subset G'^d$ in \mathcal{G}' . By abuse of notation we will also consider each of s and s' acting just on the diagonal. If $|s'(H_0^d)| = 1$, by construction of \mathcal{G}'' , we have $|s'(H^d)| = 1$. Indeed, this was the property we noted in Lemma 9. By Lemma 1, this would mean s' is uniformly mapping \mathcal{H} to one vertex, which is impossible as s' is surjective. Now we will work exclusively in the diagonal copy G'^d . As $1 < |s'(H_0^d)| < m$ is not possible either due to Lemma 7, we find that $|s'(H_0^d)| = m$, and indeed s' maps H_0^d to a copy of itself in H which we will call $H'_0 = i(H_0^d)$ for some isomorphism i .

We claim that $\text{Spill}_m(H[H'_0, i(\text{HC}_0^d)]) = V(H)$. In order to see this, consider a vertex $y \in V(H)$. As s' is surjective, there exists a vertex $x \in V(\mathcal{H})$ with $s'(x) = y$. By construction, x belongs to some top copy of DC_m^* in Cyl_m^* in $F(H_0, \text{HC}_0)$. We can extend i^{-1} to an isomorphism from the copy of Cyl_m^* (which has $i(\text{HC}_0^d)$ as its bottom cycle) in the graph $F(H'_0, i(\text{HC}_0^d))$ to the copy of Cyl_m^* (which has HC_0^d as its bottom cycle) in the graph $F(H_0, \text{HC}_0)$. We define a mapping r^* from $F(H'_0, i(\text{HC}_0^d))$ to H by $r^*(u) = s' \circ i^{-1}(u)$ if



■ **Figure 3** An interesting tournament H on six vertices (self-loops are not drawn). This tournament does not retract to the DC_3^* on the left-hand side, yet $\text{Spill}_3(H[DC_3^*, DC_3]) = V(H)$.

u is on the copy of Cyl_m^* in $F(H'_0, i(\text{HC}_0^d))$ and $r^*(u) = u$ otherwise. We observe that $r^*(u) = u$ if $u \in V(H'_0)$ as s' coincides with i on H_0 . As H_0^d separates the other vertices of the copy of Cyl_m^* from $V(H^d) \setminus V(H_0^d)$, in the sense that removing H_0^d would disconnect them, this means that r^* is a retraction from $F(H'_0, i(\text{HC}_0^d))$ to H . We find that r^* maps $i(x)$ to $s' \circ i^{-1}(i(x)) = s'(x) = y$. Moreover, as x is in the top copy of DC_m^* in $F(H_0, \text{HC}_0)$, we conclude that y always belongs to $\text{Spill}_m(H[H'_0, i(\text{HC}_0^d)])$.

As $\text{Spill}_m(H[H'_0, i(\text{HC}_0^d)]) = V(H)$, we find, by assumption of the lemma, that there exists a retraction r from H to H'_0 . Now, recalling that we can view s' acting just on the diagonal copy H^d of H , $i^{-1} \circ r \circ s'$ is the desired retraction of G to H_0 . ◀

We now need to deal with the situation in which we have an isomorphic copy $H'_0 = i(H_0)$ of H_0 in H with $\text{Spill}_m(H[H'_0, i(\text{HC}_0)]) = V(H)$, such that H does not retract to H'_0 (see Figure 3 for an example). We cannot deal with this case in a direct manner and first show another base case. For this we need the following lemma and an extension of endo-triviality that we discuss afterwards.

► **Lemma 13** ([15]). *Let H be a reflexive tournament, containing a subtournament H_0 so that any endomorphism of H that fixes H_0 as a graph is an automorphism. Then any endomorphism of H that maps H_0 to an isomorphic copy $H'_0 = i(H_0)$ of itself is an automorphism of H .*

Let H_0 be an induced subgraph of a digraph H . We say that the pair (H, H_0) is *endo-trivial* if all endomorphisms of H that fix H_0 are automorphisms.

► **Lemma 14** (Base Case II). *Let H be a reflexive tournament with a subtournament H_0 with Hamilton cycle HC_0 so that (H, H_0) and H_0 are endo-trivial and $\text{Spill}_m(H[H_0, \text{HC}_0]) = V(H)$. Then H -RETRACTION can be polynomially reduced to QCSP(H).*

Proof. Let G be an instance of H -RETRACTION. Let m be the size of $|V(H_0)|$ and n be the size of $|V(H)|$. We build an instance φ of QCSP(H) in the following fashion. Consider all possible functions $\lambda : [n] \rightarrow V(H)$. For some such λ , let $\mathcal{G}(\lambda)$ be the graph enriched with constants c_1, \dots, c_n where these are interpreted over some subset of $V(H)$ according to λ in the natural way (acting on the subscripts).

Let $\mathcal{G} = \bigotimes_{\lambda \in V(H)^{[n]}} \mathcal{G}(\lambda)$. Let G^d , H^d and H_0^d be the diagonal copies of G , H and H_0 in \mathcal{G} . Let \mathcal{H} be the subgraph of \mathcal{G} induced by $V(H) \times \dots \times V(H)$. Note that the constants c_1, \dots, c_n live in \mathcal{H} . Now build \mathcal{G}' from \mathcal{G} by augmenting a new copy of Cyl_m^* for every vertex $v \in V(\mathcal{H}) \setminus V(H_0^d)$. Vertex v is to be identified with any vertex in the top copy of DC_m^* in Cyl_m^* and the bottom copy of DC_m^* is to be identified with HC_0 in H_0^d according to the identity function.

Finally, build φ from the canonical query of \mathcal{G}' where we additionally turn the constants c_1, \dots, c_n to outermost universal variables.

First suppose that G retracts to H by r . Let λ be some assignment of the universal variables of φ to H . To prove $\varphi \in \text{QCSP}(H)$ it suffices to prove that there is a homomorphism from \mathcal{G}' to H that extends λ and for this it suffices to prove that there is a homomorphism from \mathcal{G} that extends λ . This is always possible since we have $\text{Spill}_m(H[H_0, \text{HC}_0]) = V(H)$ by assumption.

Henceforth let us consider the homomorphic image of \mathcal{G} that is $\mathcal{G}(\lambda)$. To prove $\varphi \in \text{QCSP}(H)$ it suffices to prove that there is a homomorphism from $G(\lambda)$ to H that extends λ . Note that it will be sufficient to prove that G retracts to H . Well this was our original assumption so we are done.

Suppose now $\varphi \in \text{QCSP}(H)$. Choose some surjection for λ , the assignment of the universal variables of φ to H . Recall $N = |V(H)^{[n]}|$. The evaluation of the existential variables that witness $\varphi \in \text{QCSP}(H)$ induces a surjective homomorphism s from \mathcal{G}' to H which contains within it a surjective homomorphism s' from $\mathcal{H} = H^N$ to H . Consider the diagonal copy of $H_0^d \subset H^d \subset G^d$ in $(G)^N$. By abuse of notation we will also consider each of s and s' acting just on the diagonal. If $|s'(H_0^d)| = 1$, by construction of \mathcal{G}' , we have $|s'(H^d)| = 1$. By Lemma 1, this would mean s' is uniformly mapping \mathcal{H} to one vertex, which is impossible as s' is surjective. Now we will work exclusively on the diagonal copy G^d . As $1 < |s'(H_0^d)| < m$ is not possible either due to Lemma 7, we find that $|s'(H_0^d)| = m$, and indeed s' maps H_0^d to a copy of itself in H which we will call $H'_0 = i(H_0^d)$ for some isomorphism i .

As (H, H_0) is endo-trivial, Lemma 13 tells us that the restriction of s' to H^d is an automorphism of H^d , which we call α . The required retraction from G to H is now given by $\alpha^{-1} \circ s'$. \blacktriangleleft

3.3 The strongly connected case: Generalising the Base Cases

We now generalise the two base cases to more general cases via some recursive procedure. Afterwards we will show how to combine these two cases to complete our proof. We will first need a slightly generalised version of Lemma 13, which nonetheless has virtually the same proof.

► **Lemma 15** ([15]). *Let $H_2 \supset H_1 \supset H_0$ be a sequence of strongly connected reflexive tournaments, each one a subtournament of the one before. Suppose that any endomorphism of H_1 that fixes H_0 is an automorphism. Then any endomorphism h of H_2 that maps H_0 to an isomorphic copy $H'_0 = i(H_0)$ of itself also gives an isomorphic copy of H_1 in $h(H_1)$.*

The following two lemmas generalise Lemmas 12 and 14. The proof of the second is omitted.

► **Lemma 16** (General Case I). *Let $H_0, H_1, \dots, H_k, H_{k+1}$ be reflexive tournaments, the first k of which have Hamilton cycles $\text{HC}_0, \text{HC}_1, \dots, \text{HC}_k$, respectively, so that $H_0 \subseteq H_1 \subseteq \dots \subseteq H_k \subseteq H_{k+1}$. Assume that $H_0, (H_1, H_0), \dots, (H_k, H_{k-1})$ are endo-trivial and that*

$$\begin{aligned} \text{Spill}_{a_0}(H_1[H_0, \text{HC}_0]) &= V(H_1) \\ \text{Spill}_{a_1}(H_2[H_1, \text{HC}_1]) &= V(H_2) \\ \vdots &\quad \quad \quad \vdots \quad \quad \quad \vdots \\ \text{Spill}_{a_{k-1}}(H_k[H_{k-1}, \text{HC}_{k-1}]) &= V(H_k). \end{aligned}$$

Moreover, assume that H_{k+1} retracts to H_k and also to every isomorphic copy $H'_k = i(H_k)$ of H_k in H_{k+1} with $\text{Spill}_{a_k}(H_{k+1}[H'_k, i(\text{HC}_k)]) = V(H_{k+1})$. Then H_k -RETRACTION can be polynomially reduced to $\text{QCSP}(H_{k+1})$.

Proof. Let a_{k+1}, \dots, a_0 be the cardinalities of $|V(H_{k+1})|, \dots, |V(H_0)|$, respectively. Let $n = a_{k+1}$. Let G be an instance of H_k -RETRACTION. We will build an instance φ of $\text{QCSP}(H_{k+1})$ in the following fashion. First, take a copy of H_{k+1} together with G and build G' by identifying these on the copy of H_k that they both possess as an induced subgraph.

Consider all possible functions $\lambda : [n] \rightarrow V(H_{k+1})$. For some such λ , let $\mathcal{G}'(\lambda)$ be the graph enriched with constants c_1, \dots, c_n where these are interpreted over some subset of $V(H_{k+1})$ according to λ in the natural way (acting on the subscripts).

Let $\mathcal{G}' = \bigotimes_{\lambda \in V(H_{k+1})^{[n]}} \mathcal{G}'(\lambda)$. Let G'^d, H_{k+1}^d and H_k^d etc. be the diagonal copies of G'^d, H_{k+1} and H_k in \mathcal{G}' . Let \mathcal{H}_{k+1} be the subgraph of \mathcal{G}' induced by $V(H_{k+1}) \times \dots \times V(H_{k+1})$. Note that the constants c_1, \dots, c_n live in \mathcal{H}_{k+1} . Now build \mathcal{G}'' from \mathcal{G}' by augmenting a new copy of $\text{Cyl}_{a_k}^*$ for every vertex $v \in V(\mathcal{H}_{k+1}) \setminus V(H_k^d)$. Vertex v is to be identified with any vertex in the top copy of DC_{a_k} in $\text{Cyl}_{a_k}^*$ and the bottom copy of DC_{a_k} is to be identified with HC_k in H_k^d according to the identity function.

Then, for each $i \in [k]$, and $v \in V(H_i^d) \setminus V(H_{i-1}^d)$, add a copy of $\text{Cyl}_{a_{i-1}}^*$, where v is identified with any vertex in the top copy of $\text{DC}_{a_{i-1}}$ in $\text{Cyl}_{a_{i-1}}^*$ and the bottom copy of $\text{DC}_{a_{i-1}}^*$ is to be identified with H_{i-1} according to the identity map of $\text{DC}_{a_{i-1}}^*$ to HC_{i-1} .

Finally, build φ from the canonical query of \mathcal{G}'' where we additionally turn the constants c_1, \dots, c_n to outermost universal variables.

First suppose that G retracts to H_k . Let λ be some assignment of the universal variables of φ to H_{k+1} . To prove $\varphi \in \text{QCSP}(H_{k+1})$ it suffices to prove that there is a homomorphism from \mathcal{G}'' to H_{k+1} that extends λ and for this it suffices to prove that there is a homomorphism from \mathcal{G}' that extends λ . Let us explain why. We map the various copies of $\text{Cyl}_{a_{i-1}}^*$ in \mathcal{G}'' in any suitable fashion, which will always exist due to our assumptions and the fact that $\text{Spill}_{a_k}(H_{k+1}[H_k, \text{HC}_k]) = V(H_{k+1})$, which follows from our assumption that H_{k+1} retracts to H_k and Lemma 11.

Henceforth let us consider the homomorphic image of \mathcal{G}' that is $\mathcal{G}'(\lambda)$. To prove $\varphi \in \text{QCSP}(H_{k+1})$ it suffices to prove that there is a homomorphism from $\mathcal{G}'(\lambda)$ to H_{k+1} that extends λ . Note that it will be sufficient to prove that G' retracts to H_{k+1} . Let h be the natural retraction from G' to H_{k+1} that extends the known retraction from G to H_k . We are done.

Suppose now $\varphi \in \text{QCSP}(H_{k+1})$. Choose some surjection for λ , the assignment of the universal variables of φ to H_{k+1} . Let $N = |V(H_{k+1})^{[n]}|$. The evaluation of the existential variables that witness $\varphi \in \text{QCSP}(H_{k+1})$ induces a surjective homomorphism s from \mathcal{G}' to H_{k+1} which contains within it a surjective homomorphism s' from $\mathcal{H} = H_{k+1}^N$ to H_{k+1} . Consider the diagonal copy of $H_0^d \subset \dots \subset H_k^d \subset H_{k+1}^d \subset G'^d$ in \mathcal{G}' . By abuse of notation we will also consider each of s and s' acting just on the diagonal. If $|s'(H_0^d)| = 1$, by construction of \mathcal{G}'' , we could follow the chain of spills to deduce that $|s'(H_{k+1}^d)| = 1$, which is not possible by Lemma 1. Moreover, $1 < |s'(H_0^d)| < |V(H_0^d)|$ is impossible due to Lemma 7. Now we will work exclusively on the diagonal copy G'^d .

Thus, $|s'(H_0^d)| = |V(H_0^d)|$ and indeed s' maps H_0^d to an isomorphic copy of itself in H_{k+1} which we will call $H'_0 = i(H_0^d)$. We now apply Lemma 15 as well as our assumed endotrivialities to derive that s' in fact maps H_k^d by the isomorphism i to a copy of itself in H_{k+1} which we will call H'_k . Since s' is surjective, we can deduce that $\text{Spill}_{a_k}(H_{k+1}[H'_k, i(\text{HC}_k^d)]) = V(H_{k+1})$ in the same way as in the proof of Lemma 12. and so there exists a retraction r from H_{k+1} to H'_k . Now $i^{-1} \circ r \circ s'$ gives the desired retraction of G to H_k . \blacktriangleleft

► **Lemma 17** (General Case II). *Let $H_0, H_1, \dots, H_k, H_{k+1}$ be reflexive tournaments, the first $k+1$ of which have Hamilton cycles $\text{HC}_0, \text{HC}_1, \dots, \text{HC}_k$, respectively, so that $H_0 \subseteq H_1 \subseteq \dots \subseteq H_k \subseteq H_{k+1}$. Suppose that $H_0, (H_1, H_0), \dots, (H_k, H_{k-1}), (H_{k+1}, H_k)$ are endo-trivial*

and that

$$\begin{array}{ll}
\text{Spill}_{a_0}(H_1[H_0, HC_0]) & = V(H_1) \\
\text{Spill}_{a_1}(H_2[H_1, HC_1]) & = V(H_2) \\
\vdots & \vdots \quad \vdots \\
\text{Spill}_{a_{k-1}}(H_k[H_{k-1}, HC_{k-1}]) & = V(H_k) \\
\text{Spill}_{a_k}(H_{k+1}[H_k, HC_k]) & = V(H_{k+1})
\end{array}$$

Then H_{k+1} -RETRACTION can be polynomially reduced to QCSP(H_{k+1}).

► **Corollary 18.** *Let H be a non-trivial strongly connected reflexive tournament. Then QCSP(H) is NP-hard.*

Proof. As H is a strongly connected reflexive tournament, which has more than one vertex by our assumption, H is not transitive. Note that H -RETRACTION is NP-complete (see Section 4.5 in [15], using results from [14, 5, 16]). Thus, if H is endo-trivial, the result follows from Lemma 12 (note that we could also have used Corollary 8).

Suppose H is not endo-trivial. Then, by Lemma 4, H is not retract-trivial either. This means that H has a non-trivial retraction to some subtournament H_0 . We may assume that H_0 is endo-trivial, as otherwise we will repeat the argument until we find a retraction from H to an endo-trivial (and consequently strongly connected) subtournament.

Suppose that H retracts to all isomorphic copies $H'_0 = i(H_0)$ of H_0 within it, except possibly those for which $\text{Spill}_m(H[H'_0, i(HC_0)]) \neq V(H)$. Then the result follows from Lemma 12. So there is a copy $H'_0 = i(H_0)$ to which H does not retract for which $\text{Spill}_m(H[H'_0, i(HC_0)]) = V(H)$. If (H, H'_0) is endo-trivial, the result follows from Lemma 14. Thus we assume (H, H'_0) is not endo-trivial and we deduce the existence of $H'_0 \subset H_1 \subset H$ (H_1 is strictly between H and H'_0) so that (H_1, H'_0) and H'_0 are endo-trivial and H retracts to H_1 . Now we are ready to break out. Either H retracts to all isomorphic copies of $H'_1 = i(H_1)$ in H , except possibly for those so that $\text{Spill}_m(H[H'_1, i(HC_1)]) \neq V(H)$, and we apply Lemma 16, or there exists a copy H'_1 , with $\text{Spill}_m(H[H'_1, i(HC_1)]) = V(H)$, to which it does not retract. If (H, H'_1) is endo-trivial, the result follows from Lemma 17. Otherwise we iterate the method, which will terminate because our structures are getting strictly larger. ◀

3.4 An initial strongly connected component that is non-trivial

This section follows a similar methodology to the previous two sections. However, the proofs are a little more involved and are omitted from this version of the paper.

► **Corollary 19.** *Let H be a reflexive tournament with an initial strongly connected component that is non-trivial. Then QCSP(H) is NP-hard.*

4 The Proof of the NL Cases of the Dichotomy

A particular role in the tractable part of our dichotomy will be played by TT_2^* , the reflexive transitive 2-tournament, which has vertex set $\{0, 1\}$ and edge set $\{(0, 0), (0, 1), (1, 1)\}$.

► **Lemma 20.** *Let $H = H_1 \Rightarrow \dots \Rightarrow H_n$ be a reflexive tournament on $m + 2$ vertices with $V(H_1) = \{s\}$ and $V(H_n) = \{t\}$. Then there exists a surjective homomorphism from $(\text{TT}_2^*)^m$ to H .*

Proof. Build a surjective homomorphism f from $(\text{TT}_2^*)^m$ to H in the following fashion. Let \bar{x}_i be the m -tuple which has 1 in the i th position and 0 in all other positions. For $i \in [m]$, let f map \bar{x}_i to i . Let f map $(0, \dots, 0)$ to s and everything remaining to t .

By construction, f is surjective. To see that f is a homomorphism, let $((y_1, \dots, y_m), (z_1, \dots, z_m)) \in E((\text{TT}_2^*)^m)$, which is the case exactly when $y_i \leq z_i$ for all $i \in [m]$. Let $f(y_1, \dots, y_m) = u$ and $f(z_1, \dots, z_m) = v$. First suppose that y_1, \dots, y_m are all 0. Then $u = s$. As s has an out-edge to every vertex of H , we find that $(u, v) \in E(H)$. Now suppose that y_1, \dots, y_m contains a single 1. If $(y_1, \dots, y_m) = (z_1, \dots, z_m)$, then $u = v$. As H is reflexive, we find that $(u, v) \in E(H)$. If $(y_1, \dots, y_m) \neq (z_1, \dots, z_m)$, then $v = t$. As t has an in-edge from every vertex of H , we find that $(u, v) \in E(H)$. Finally suppose that y_1, \dots, y_m contains more than one 1. Then $u = v = t$. As H is reflexive, we find that $(u, v) \in E(H)$. ◀

We also need the following lemma, which follows from combining some known results.

► **Lemma 21.** *If H is a transitive reflexive tournament then $\text{QCSP}(H)$ is in NL.*

Proof. It is noted in [15] that H has the ternary median operation as a polymorphism. It follows from well-known results (e.g. in [7, 9]) that $\text{QCSP}(H)$ is in NL. ◀

The other tractable cases are more interesting.

We are now ready to prove the main result of this section.

► **Theorem 22.** *Let $H = H_1 \Rightarrow \dots \Rightarrow H_n$ be a reflexive tournament. If $|V(H_1)| = |V(H_n)| = 1$, then $\text{QCSP}(H)$ is in NL.*

Proof. Let $|V(H)| = m + 2$ for some $m \geq 0$. By Lemma 20, there exists a surjective homomorphism from $(\text{TT}_2^*)^m$ to H . There exists also a surjective homomorphism from H to TT_2^* ; we map s to 0 and all other vertices of H to 1. It follows from [8] that $\text{QCSP}(H) = \text{QCSP}(\text{TT}_2^*)$ meaning we may consider the latter problem. We note that TT_2^* is a transitive reflexive tournament. Hence, we may apply Lemma 21. ◀

5 Final result and remarks

We are now in a position to prove our main dichotomy theorem.

► **Theorem 23.** *Let $H = H_1 \Rightarrow \dots \Rightarrow H_n$ be a reflexive tournament. If $|V(H_1)| = |V(H_n)| = 1$, then $\text{QCSP}(H)$ is in NL; otherwise it is NP-hard.*

Proof. The NL case follow from Theorem 22. The NP-hard cases follow from Corollary 18 and Corollary 19, bearing in mind the case with a non-trivial final strongly connected component is dual to the case with a non-trivial initial strongly connected component (map edges (x, y) to (y, x)). ◀

Theorem 23 resolved the open case in Table 1. Recall that the results for the irreflexive tournaments in this table were all proven in a more general setting, namely for irreflexive semicomplete graphs. A natural direction for future research is to determine a complexity dichotomy for QCSP and SCSP for reflexive semicomplete graphs. We leave this as an interesting open direction.

References

- 1 Jørgen Bang-Jensen, Pavol Hell, and Gary MacGillivray. The complexity of colouring by semicomplete digraphs. *SIAM Journal on Discrete Mathematics*, 1(3):281–298, 1988.
- 2 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems - a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012.
- 3 Ferdinand Börner, Andrei A. Bulatov, Hubie Chen, Peter Jeavons, and Andrei A. Krokhin. The complexity of constraint satisfaction games and QCSP. *Inf. Comput.*, 207(9):923–944, 2009. doi:10.1016/j.ic.2009.05.003.
- 4 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330, 2017.
- 5 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.
- 6 Paul Camion. Chemins et circuits hamiltoniens de graphes complets. *Comptes Rendus de l'Académie des Sciences Paris*, 249:2151–2152, 1959.
- 7 Hubie Chen. The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case. *SIAM J. Comput.*, 37(5):1674–1701, 2008. doi:10.1137/060668572.
- 8 Hubie Chen, Florent R. Madelaine, and Barnaby Martin. Quantified constraints and containment problems. *Logical Methods in Computer Science*, 11(3), 2015. Extended abstract appeared at LICS 2008. This journal version incorporates principal part of CP 2012 *Containment, Equivalence and Coreness from CSP to QCSP and Beyond*. doi:10.2168/LMCS-11(3:9)2015.
- 9 Víctor Dalmau and Andrei A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008.
- 10 Petar Dapic, Petar Markovic, and Barnaby Martin. Quantified constraint satisfaction problem on semicomplete digraphs. *ACM Trans. Comput. Log.*, 18(1):2:1–2:47, 2017. doi:10.1145/3007899.
- 11 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.
- 12 Petr A. Golovach, Daniël Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.
- 13 P. G. Kolaitis and M. Y. Vardi. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*, chapter A logical Approach to Constraint Satisfaction, pages 339–370. Springer-Verlag New York, Inc., 2005.
- 14 Benoit Larose. Taylor operations on finite reflexive structures. *International Journal of Mathematics and Computer Science*, 1(1):1–26, 2006.
- 15 Benoit Larose, Barnaby Martin, and Daniël Paulusma. Surjective H-colouring over reflexive digraphs. *TOCT*, 11(1):3:1–3:21, 2019. doi:10.1145/3282431.
- 16 Benoit Larose and László Zádori. Finite posets and topological spaces in locally finite varieties. *Algebra Universalis*, 52(2):119–136, 2005.
- 17 Barnaby Martin. QCSP on partially reflexive forests. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, pages 546–560, 2011. doi:10.1007/978-3-642-23786-7_42.
- 18 Barnaby Martin and Florent R. Madelaine. Towards a trichotomy for quantified H-coloring. In *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, pages 342–352, 2006. doi:10.1007/11780342_36.
- 19 Narayan Vikas. Algorithms for partition of some class of graphs under compaction and vertex-compaction. *Algorithmica*, 67(2):180–206, 2013.

- 20 Narayan Vikas. Computational complexity of graph partition under vertex-compaction to an irreflexive hexagon. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 69:1–69:14, 2017.
- 21 Alexander Wires. Dichotomy for finite tournaments of mixed-type. *Discrete Mathematics*, 338(12):2523–2538, 2015. doi:10.1016/j.disc.2015.06.024.
- 22 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342, 2017.
- 23 Dmitriy Zhuk. No-rainbow problem and the surjective constraint satisfaction problem, 2020. To appear LICS 2021. arXiv:2003.11764.
- 24 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.
- 25 Dmitriy Zhuk and Barnaby Martin. QCSP monsters and the demise of the Chen Conjecture. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 91–104. ACM, 2020. doi:10.1145/3357713.3384232.

Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing

Thomas Lavastida ✉

Carnegie Mellon University, Pittsburgh, PA, USA

Benjamin Moseley ✉

Carnegie Mellon University, Pittsburgh, PA, USA

R. Ravi ✉

Carnegie Mellon University, Pittsburgh, PA, USA

Chenyang Xu¹ ✉

Zhejiang University, China

Abstract

We propose a new model for augmenting algorithms with predictions by requiring that they are formally learnable and instance robust. Learnability ensures that predictions can be efficiently constructed from a reasonable amount of past data. Instance robustness ensures that the prediction is robust to modest changes in the problem input, where the measure of the change may be problem specific. Instance robustness insists on a smooth degradation in performance as a function of the change. Ideally, the performance is never worse than worst-case bounds. This also allows predictions to be objectively compared.

We design online algorithms with predictions for a network flow allocation problem and restricted assignment makespan minimization. For both problems, two key properties are established: high quality predictions can be learned from a small sample of prior instances and these predictions are robust to errors that smoothly degrade as the underlying problem instance changes.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Learning-augmented algorithms, Online algorithms, Flow allocation

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.59

Related Version *Full Version*: <https://arxiv.org/abs/2011.11743>

Funding *Chenyang Xu*: Supported in part by Science and Technology Innovation 2030 – “The Next Generation of Artificial Intelligence” Major Project No.2018AAA0100902 and China Scholarship Council No.201906320329.

Thomas Lavastida, and Benjamin Moseley: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.

R. Ravi: Supported in part by the U. S. Office of Naval Research under award number N00014-21-1-2243 and the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

1 Introduction

Inspired by advances in machine learning, there is an interest in augmenting algorithms with predictions, especially in online algorithm design [21, 10, 12, 15, 26, 28, 34, 27]. Algorithms augmented with predictions have had empirical success in domains such as look-up tables [26], caching [28], and bloom-filters [31]. These successes and the availability of data to make predictions using machine learning have motivated the development of new analysis models

¹ The corresponding author



© Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 59; pp. 59:1–59:17

Leibniz International Proceedings in Informatics



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

for going beyond worst-case bounds where an algorithm is supplied with accurate predictions. In these models, an algorithm is given access to a prediction about the problem instance. The algorithm's performance is bounded in terms of the quality of this prediction. Typically the algorithm learns such predictions from a limited amount of past data leading to *error-prone* predictions. The algorithm with accurate predictions should result in better performance than the best worst-case bound. Ideally, the algorithm never performs asymptotically worse than the best worst-case algorithm even if the prediction error is large. In-between, there is a graceful degradation in performance as the prediction error increases. For example, competitive ratio or running time can be parameterized by prediction error. See [32] for a survey.

Learnable and Instance Robust Predictions. The model proposed in this paper has two pillars for augmenting algorithms with predictions. (**Learnability**;) Predictions should be learnable from representative data. (**Instance Robustness**²;) Predictions should be robust to minor changes in the problem instance. As in prior models, determining what to predict remains a key algorithmic challenge.

Suppose there is an unknown distribution \mathcal{D} over instances \mathcal{I} . Building on data driven algorithm design [21] and PAC-learning models, we require that predicted parameters are provably learnable using a small number of sample instances from \mathcal{D} . The sample complexity of this task can be used to compare how difficult different predictions are to construct. Practically, the motivation is that parameters are learned from prior data (e.g. instances of the problem).

In practice, future problem instances may not come from the same distribution used to learn the parameters. Therefore, we also desire predictions that are robust to modest changes in the input. In particular, if the predictions perform well on some instance \mathcal{I} , then the performance on a nearby instance \mathcal{I}' should be bounded as a function of the distance between these instances. This measure of the distance between instances is necessarily problem specific.

We note that learnability is rarely addressed in prior work and our robustness model differs from many prior works by bounding the error by differences in problem instances (instance robustness), rather than by the differences in the predictions themselves (parameter robustness). We present learnable and instance-robust predictions for two concrete online problems.

Online Flow Allocation Problem. We consider a general flow and matching problem. The input is a Directed-Acyclic-Graph (DAG) G . Each node v has an associated capacity C_v . There is a sink node t , such that all nodes in the DAG can reach the sink. Online source nodes arrive that have no incoming edges (and never have any incoming edges in the future) and the other nodes are offline and fixed. We will refer to online nodes I as impressions. When impression i arrives, it is connected to a subset of nodes N_i . At arrival, the algorithm must decide a (fractional) flow from i to t of value at most 1 obeying the node capacities. This flow is fixed the moment the node arrives. The goal is to maximize the total flow that reaches t without exceeding node capacities. Instances are defined by the number of each *type* of impression. The type of an impression is given by the subset of the nodes of G to which it has outgoing arcs. We may consider specific worst-case instances or a distribution over types

² Note that the robustness here is different from the definition of robustness mentioned in previous work, which we refer to as parameter robustness. See Section 2 for a discussion.

in our analysis. This problem captures fractional versions of combinatorial problems such as online matching, unweighted Adwords, and finding a maximum independent set in a laminar matroid or gammoid. We call the problem the Online Flow Allocation Problem.

Restricted Assignment Makespan Minimization. In this problem, there are m machines and n jobs arrive in an online order. When job j arrives it must be immediately and irrevocably assigned to a machine. The job has size p_j and can only be assigned to a subset of machines specific to that job. After j is assigned the next job arrives. A machine's load is the total size of all jobs assigned to it. The goal is to minimize the makespan, or maximum load, of the assignment.

1.1 Overview of Results for Flow Allocation and Restricted Assignment

We first focus on the flow allocation problem and then we give similar results for the makespan minimization problem.

Node Parameters. Our results on learnability and robustness are enabled by showing the existence of node weights which capture interesting combinatorial properties of flows. Inspired by the weights proven in [2] for bipartite matching, we establish that there is a single weight for each node in the DAG that completely describe near optimal flows on a single problem instance. The weights determine an allocation of flow for each impression which is independent of the other impressions. Each node in the DAG routes the flow leaving it proportionally to the weights of its outgoing neighbors. Moreover, the flow is near optimal, giving a $(1 - \epsilon)$ -approximate solution for any constant $\epsilon > 0$ (but requiring time polynomial in $1/\epsilon$ to compute). Given these weights, the flow can be computed in one forward pass for each impression in isolation. Thus they can be used online if given as a prediction. These weights are also efficiently computable offline given the entire problem instance (see Theorem 3).

Instance Robustness. We measure the distance of the two instances as the difference of the number of impressions of each type (see Theorem 5). We show that if the weights are near optimal for one instance, the performance degrades gracefully according to the distance between the two instances. This distance is defined for any two instances irrespective of whether they are generated by specific distributions³.

Learnability. For learnability it is assumed that impressions are drawn from an unknown distribution over types. We show that learning near-optimal weights for this distribution has low sample complexity under two assumptions. First, we assume the unknown distribution is a product distribution. Second, we assume that the optimal solution of the “expected instance” (to be defined later) has at least a constant amount of flow routed through each node. In the 2-layer case, this assumption can be simplified to requiring each node's capacity to be at least a constant (depending on $\frac{1}{\epsilon}$).⁴ The number of samples is polynomial in the size of the DAG without the impressions. Note that in problems such as Adwords, the impressions are usually much larger than the fixed portion of the graph.

We now present our main theorem on the flow allocation problem.

³ We also show that our predictions for the online flow allocation problem have “parameter robustness”, the kind of robustness that has been considered in prior work (Theorem 6).

⁴ This is similar to the lower bound requirement on budgets in the online analysis of the AdWords problem [16].

► **Theorem 1** (Flow Allocation - Informal). *There exist algorithmic parameters for the Online Flow Allocation problem with the following properties:*

- (i) *(Learnability) Learning near-optimal parameters has sample complexity polynomial in $\frac{1}{\epsilon}$ and the size of the graph excluding the impressions. These parameters result in an online algorithm that is a $(1 - \epsilon)$ -approximate solution in expectation as compared to the expected optimal value on the distribution for any constant $\epsilon > 0$. (Theorem 4)*
- (ii) *(Instance Robustness) Using the optimal parameters for an instance on another instance gives a competitive ratio that improves as their distance decreases, where the distance is proportional to the difference of impressions (Theorem 5).*
- (iii) *(Worst-Case Robustness) The competitive ratio of the online algorithm using the parameters is never worse than $\frac{1}{d+1}$, regardless of the distance between the two instances, where d is the diameter of G . (Theorem 5)*

The theorem states that weights are learnable and only a small number of samples are required to construct weights that are near optimal. These predictions break the worst-case $1 - \frac{1}{e}$ bound on the competitive ratio for any randomized algorithm for online fractional matching, a special case. Moreover, the difference in the types of impressions between two instances gives a metric under which we can demonstrate instance robustness. Further the algorithm has worst-case guarantees, i.e. the ratio is never worse than $\frac{1}{d+1}$, which is tight for deterministic integral online algorithms and d -layer graphs (see Theorem 61 in the full version) even though we output fractional allocations.

We now discuss our results for makespan minimization.

► **Theorem 2** (Restricted Assignment - Informal). *There exist algorithmic parameters for the Restricted Assignment Makespan Minimization problem with the following properties:*

- (i) *(Learnability) Learning the near optimal parameters has sample complexity polynomial in m , the number of machines, and $\frac{1}{\epsilon}$. These parameters result in an online algorithm that is a $(1 + \epsilon)$ approximate solution in expectation as compared to the expected optimal value on the distribution for any constant $\epsilon > 0$. (Theorem 8)*
- (ii) *(Instance Robustness) Using the optimal parameters for any instance on a nearby instance gives a competitive ratio for fractional assignment that is proportional to their distance, where the distance is proportional to the relative difference of job sizes of the same type. (Theorem 7)*
- (iii) *(Worst-Case Robustness) The competitive ratio of the algorithm using the parameters is never worse than $O(\log m)$, matching the known $\Omega(\log m)$ lower-bound on any integral online algorithm. (Theorem 7)*

This theorem shows that the predictions of [27] have much stronger properties than what is known and are learnable and instance robust. That paper left open the question if their predictions can be formally learned in any model. Moreover, it was not known if they are instance robust. We remark that this theorem assumes fractional assignments, whereas the original problem (and the lower bound [8]) requires integer assignments. Lattanzi et al. [27] shows that any fractional assignment can be rounded online while losing a $O((\log \log m)^3)$ factor in the makespan.

1.2 Related Work

Algorithms with Predictions. In this paper, we consider augmenting the standard model of online algorithms with erroneous predictions. Several online problems have been studied in this context, including caching [28, 35, 23, 37], page migration [22], metrical task systems [6], ski rental [34, 19, 3], scheduling [34], load balancing [27], online linear optimization [13], speed scaling [38], set cover [39], and bipartite matching and secretary problems [7].

Antoniadis et al. [7] studies online weighted bipartite matching problems with predictions. The main aspect of this work which distinguishes it from ours is that it considers the random order arrival model, rather than adversarial orders.

Mahdian et al. [29] focuses on the design of robust algorithms. Rather than considering online algorithms which use a prediction, they consider two black box online algorithms, one optimistic and the other pessimistic. The goal is to give an online algorithm which never performs much worse than the better of these two algorithms for any given instance. This is shown for problems such as load balancing, facility location, and ad allocation.

The predictions utilized in our algorithm come in the form of vertex weights that guide a proportional allocation scheme. Agrawal et al. [2] first studied proportional allocations for maximum cardinality fractional matching as well as weighted fractional matchings with high entropy. Lattanzi et al. [27] utilize similar predictions based on proportional weights to give algorithms with predictions for online load balancing.

Data-Driven Algorithm Design. This paper considers the learnability of the predictions through the model of data-driven algorithms. In classical algorithm design, the main desire is finding an algorithm that performs well in the worst case against all inputs for some measure of performance, e.g. running time or space usage. Data-driven algorithm design [21, 9, 11, 10, 12, 15], in contrast, wants to find an algorithm that performs well on the instances that the user is typically going to see in practice. This is usually formalized by fixing a class of algorithms and an unknown distribution over instances, capturing the idea that some (possibly worst case) instances are unlikely to be seen in practice. The typical question asked is: how many sample instances are needed to guarantee you have found the best algorithm for your application domain?

Other Related Work. Online matching and related allocation problems have been extensively studied in both the adversarial arrival setting [25, 24, 17, 30] and with stochastic arrivals [16, 18, 20, 33, 1]. A related but different setting to ours is the online algorithms with advice setting [14]. Here the algorithm has access to an oracle which knows the offline input. The oracle is allowed to communicate information to the algorithm about the full input, and the goal is to understand how many bits of information are necessary to achieve a certain competitive ratio. This has also been extended to the case where the advice can be arbitrarily wrong [4]. This can be seen as similar to our model, however the emphasis isn't on tying the competitive ratio to the amount of error in the advice.

2 Algorithms with Learnable and Instance-Robust Predictions

Learnability via Sample Complexity. We consider the following setup inspired by PAC learning and recently considered in data-driven algorithms. Assume a maximization problem and let \mathcal{D} be an unknown distribution over problem instances. Let $\text{ALG}(\mathcal{I}, y)$ be the performance⁵ of an algorithm using parameters y on instance \mathcal{I} . The ideal prediction for this distribution is then $y^* := \arg \max_y \mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[\text{ALG}(\mathcal{I}, y)]$. Since we assume that \mathcal{D} is unknown, we wish to learn from samples. In particular, we wish to use some number s of independent samples to compute a parameter \hat{y} such that $\mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[\text{ALG}(\mathcal{I}, \hat{y})] \geq (1 - \epsilon) \mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[\text{ALG}(\mathcal{I}, y^*)]$

⁵ In general this can be any performance metric, such as running time or solution value. Here we focus on the value of some objective function such as the size of a fractional flow.

with probability $1 - \delta$, for any $\epsilon, \delta \in (0, 1)$. The sample complexity s depends on the problem size as well as $1/\epsilon$ and $1/\delta$. As is standard in learning theory, we require the sample complexity to be polynomial in these parameters⁶.

Inspired by competitive analysis, we also compare to the following stronger benchmark in this paper. For any instance \mathcal{I} , let $\text{OPT}(\mathcal{I})$ be the value of an optimal solution on \mathcal{I} . We give learning algorithms producing predicted parameters \hat{y} such that $\mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[\text{ALG}(\mathcal{I}, \hat{y})] \geq (1 - \epsilon)\mathbb{E}_{\mathcal{I} \sim \mathcal{D}}[\text{OPT}(\mathcal{I})]$ and polynomial sample complexity under the assumptions on \mathcal{D} described earlier. Note that this guarantee implies the first one.

Instance Robustness. Let \mathcal{I} and \mathcal{I}' be two problem instances, and consider running the algorithm on instance \mathcal{I}' with the prediction $y^*(\mathcal{I})$. We bound the performance of the algorithm as a function of the difference between these two instances. In contrast, prior work focuses on differences in the predicted parameters y^* and y' for the same instance \mathcal{I} . Moreover, it is desirable that the algorithm never performs worse than the best worst-case algorithm.

For example, in online flow allocation, we can consider an instance as a vector of types, i.e. \mathcal{I}_i is the number of impressions of type i . Then we can take the difference between the instances as $\gamma = \|\mathcal{I} - \mathcal{I}'\|_1$. Say $y^*(\mathcal{I})$ can be used to give a c -competitive algorithm on instance \mathcal{I} . Let α be the best competitive ratio achievable in the worst-case model. We desire an algorithm that is $\max\{f(c, \gamma), \alpha\}$ -competitive where f is a monotonic function depending on c and γ . We remark that the online model requires \mathcal{I}' to arrive in a worst-case order.

2.1 Putting the Model in Context

Relationship to Prior Predictions Model. The first main difference in this model as compared to prior work is *learnability*. With the notable exception of [3], prior work has introduced predictions without establishing they are learnable. Without this requirement there is no objective metric to detail if a prediction is reasonable or not. To see this shortcoming, imagine simply predicting the optimal solution for the problem instance. This is often not reasonable because the optimal solution is too complex to learn and use as a prediction. We introduce bounded sample complexity for learning predictions in our model to ensure predictions can be provably learned.

Next difference is in how to measure error. The performance of the algorithm is bounded in terms of the error in the prediction in the prior model. For example, say the algorithm is given a predicted vector $\hat{y}(\mathcal{I})$ for problem instance \mathcal{I} and the true vector that should have been predicted is $y^*(\mathcal{I})$. One can define $\eta_{\hat{y}(\mathcal{I})} = \|\hat{y}(\mathcal{I}) - y^*(\mathcal{I})\|_p$ to be the *error* in the parameters for some norm $p \geq 1$. The goal is to give an algorithm that is $f(\eta_{\hat{y}(\mathcal{I})})$ -competitive for an online algorithm where f is some non-decreasing function of $\eta_{\hat{y}(\mathcal{I})}$: the better the function f , the better the algorithm performance. One could also consider run time or approximation ratio similarly. Notice the bound is worst-case for a given error in the prediction. This we call *parameter robustness*.

It is perhaps more natural to define a difference between two problem instances as in our model rather than the difference between two predicted parameters. Indeed, consider predicting optimal dual linear program values. These values can be different for problem instances that are nearly identical. Therefore, accurate parameters will not be sufficient to handle inconsequential changes in the input. Instance robustness allows for more accurate

⁶ For more difficult problems, we can relax the $1 - \epsilon$ requirement to be a weaker factor.

■ **Table 1** Relationship to Prior Work.

Problem	Parameter Robustness	Learnability	Instance Robustness
Caching	[28, 35, 23, 37]	-	-
Completion Time Scheduling	[34]	-	[34]
Ski Rental	[34, 3, 19]	[3]	[34, 3, 19]
Restricted Assignment	[27]	This Paper	This Paper
b -Matching	This Paper	This Paper	This Paper
Flow Allocation	This Paper	This Paper	This Paper

comparison of two predictions on similar problem instances. More practically, instance closeness is easier to monitor than closeness of the proposed predictions to an unknown optimal prediction for the whole instance.

Learning Algorithm Parameters. Learning algorithmic parameters has distinct advantages over learning an input distribution. In many cases it can be easier to learn a decision rule than it is to learn a distribution. For example, consider the unweighted b -matching problem in bipartite graphs for large b in the online setting. In this problem there is a bipartite graph $G = (I \cup A, E)$ with capacities $b \in \mathbb{Z}_+^A$. The objective is to find a collection of edges such that each node in I is matched at most once and each node $a \in A$ is matched at most b_a times. Nodes on one side of the graph arrive online and must be matched on arrival. Say the nodes are i.i.d. sampled from an unknown discrete distribution over types. A type is defined by the neighbors of the node. Let s be the number of types. Then the sample complexity of learning the distribution is $\Omega(s)$. Notice that s could be superlinear in the number of nodes. In our results, the sample complexity is independent of the number of types in the support of the distribution. The phenomenon that it is sometimes easier to learn good algorithmic parameters rather than the underlying input distribution has been observed in several prior works. See [21, 9, 11, 10, 12, 15] for examples.

Table 1 illustrates how our paper relates to prior work, focusing on the two pillars for augmenting algorithms emphasized in our model.

Paper Organization. In this extended abstract, we give a technical overview of our results. For online flow allocation, both learnability and instance-robustness rely on showing the existence of node predictions which is described in Section 3, followed by learnability and robustness in Sections 4 and 5 respectively. Section 6 considers online load balancing and gives a brief technical overview. All proofs of our results are available in the full version of this paper.

3 Matchings and Flows: Existence of Weights

Consider a directed acyclic graph $G = (\{s, t\} \cup V, E)$, where each vertex $v \in V$ has capacity C_v and is on some $s - t$ path. Our goal is to maximize the flow sent from s to t without violating vertex capacities. Before considering the general version, we examine the 3-layered version. Say a graph is d -layered if the vertices excluding s, t can be partitioned into d ordered sets where all arcs go from one set to the next. Then the 3-layered case is defined as follows. The vertices in V are partitioned into 3 sets I, A , and B . s is connected to all of I and t is connected from all of B , while the remaining edges are only allowed to cross from I to A and from A to B . Let $N_u := \{v \in V \mid (u, v) \in E\}$ be u 's out-neighbors. We have the following result generalizing the prior work of Agrawal et al [2] on 2-layered graphs.

► **Theorem 3.** *Let $G = (\{s, t\} \cup V, E)$ be a 3-layered DAG. For each edge $(u, v) \in E$, let x_{uv} be the proportion of flow through u which is routed along (u, v) . For any $\epsilon \in (0, 1)$, there exist weights $\{\alpha_v\}_{v \in V}$ such that setting $x_{uv} = \frac{\alpha_v}{\sum_{v' \in N_u} \alpha_{v'}}$ yields a $(1 - \epsilon)$ -approximate flow. Moreover, these weights can be obtained in time $O(n^4 \log(n/\epsilon)/\epsilon^2)$.*

We can generalize this theorem to d -layered graphs. In particular, our algorithm for the d -layered case produces weights with additional properties, which we leverage to handle general DAGs. Notice that the number of weights is proportional to the number of nodes in the graph and not the number of edges. We believe it is an interesting combinatorial property that such succinct weights on the nodes can encode a good flow on the asymptotically larger number of edges and is of independent interest.

Technical Overview

Here we give a technical overview. The full proof is omitted in this version.

A Simple Algorithm for Layered Graphs. Prior work [2] showed that there exists a set of weights giving nearly the same guarantees we show, but only for bipartite graphs. The existence of such weights can be generalized to d -layer graphs easily as follows. First find an (optimal) maximum flow f . For each vertex v , let $f(v)$ be the flow going through v . Reset the vertex capacity of v to be $f(v)$. For each pair of adjacent layers find the weights between the two layers independently using the algorithm of [2], treating nodes v on the left hand side as $f(v)$ individual impressions. By the previous result, each layer only loses a negligible portion of the total flow which can be compounded to yield a low loss for these set of weights.

The above reduction does not generalize to general DAGs. One can arrange a DAG into layers, but there are fundamental algorithmic challenges with constructing weights that arise when edges cross layers. One of this paper’s algorithmic contributions is showing how to construct such weights for general DAGs. Moreover, as an intermediate result, we show how to compute the weights directly extending the approach of [2] for multi-layer graphs without first solving a flow problem optimally as in the above reduction.

Finding Weights for Bipartite Graphs. We begin by first simplifying the algorithm of [2] for bipartite graphs. Let $G = (\{s, t\} \cup I \cup A, E)$ be such a graph. In this case, the fraction of flow $u \in I$ sends to $v \in A$ simplifies to $x_{uv} = \frac{\alpha_v}{\sum_{v' \in N_u} \alpha_{v'}}$ where $\{\alpha\}_{v \in I \cup A}$ are the set of weights. Initially all of the weights are 1 for a vertex $a \in A$. Some of the nodes receive more flow than their capacity in this initial proportional allocation according to the weights. We say a node for which the current proportional allocation of flow exceeds its capacity by a $1 + \epsilon$ factor is *overallocated*. In an iteration, the algorithm decreases the weights of these nodes by a $1 + \epsilon$ factor.⁷ After this process continues for a poly-logarithmic number of iterations, we will be able to show the resulting weights result in a near optimal flow.

To prove that the final weights are near optimal, we show that the weights can be directly used to identify a vertex cut whose value matches the proportional flow given by the weights. In particular, we will partition the nodes in A based on their weight values. For a parameter β , we say a node is “above the gap” if its weight is larger than $\beta n/\epsilon$. A node of weight less

⁷ Prior work [2] performed this operation as well as *increasing* the weights of nodes whose allocation was significantly below the capacity. Our simplification to allow only decreases helps with the generalization to more complex graphs and correcting for error in the weights.

than β is below the gap. All others are in the gap. The parameter β is chosen such that the nodes in the gap contribute little to the overall flow and they can essentially be discarded via an averaging argument⁸. Assume this set is empty for simplicity. Let $\mathcal{G}(A)^+$ and $\mathcal{G}(A)^-$ be the sets of vertices in A above and below the gap, respectively.

We now describe a cut. Let $I_0 \subseteq I$ be the impression nodes adjacent to at least one node in $\mathcal{G}(A)^+$. Then the vertex cut is $I_0 \cup \mathcal{G}(A)^-$. Since all paths must either cross I_0 or $\mathcal{G}(A)^-$, this is a valid vertex cut. We now show the cut value is close to the flow achieved by the weights, completing the analysis using the weaker direction of the max-flow min-cut theorem.

First, nodes in I_0 are cut. Due to the way flow is sent based on the weight proportions, for any vertex I_0 , at least a $(1 - \epsilon)$ proportion of its flow will be sent to $\mathcal{G}(A)^+$. Since the nodes in $\mathcal{G}(A)^+$ did not decrease the weights at least once, at some point they were not over-allocated. We claim that because of this, they will never be over-allocated hence and, therefore, nodes in I_0 send nearly all of their flow to the sink successfully. Next nodes in $\mathcal{G}(A)^-$ are cut. These nodes decreased their weights (almost) every iteration because they are at or above their allocation. Thus, for all these nodes we get flow equal to their total capacity. The fraction of this flow through $\mathcal{G}(A)^-$ coming from paths using I_0 is negligible because of the weight proportions so this flow is almost disjoint from that of I_0 . Thus, we have found a proportional flow nearly matching the value of the cut identified.

General Graphs. Now we consider the more general algorithm. To convey intuition, we will only consider directly computing weights for a 3-layered graph $G = (\{s\} \cup I \cup A \cup B \cup \{t\}, E)$ where edges are between adjacent layers. This will highlight several of the new ideas. As before, weights of all nodes are initially one. And as before, a node decreases its weight if it is over-allocated, which we will refer now to as a *self-decrease*. Now though, whenever a node in A decreases its weight it does so by a $(1 + \epsilon')$ factor and those in B decrease at a $(1 + \epsilon)$ factor where $\epsilon' \leq \epsilon$.

A new challenge is that a node b in layer B may be over allocated and it may not be enough for B to reduce its weight. Indeed, B may need some neighbors in A to reduce their allocation. For instance, if $b \in B$ has neighbors in A for which it is the only neighbor, then reducing b 's weight does not change its allocation and the flow needs to be redistributed in the first layer. In this case, the nodes in B will specify that some nodes in A need to decrease their allocation. We call this a *forced decrease*. This set has to be carefully chosen and intuitively only the nodes in A that are the largest weight as compared to $b \in B$ are decreased. We run this procedure for a polylogarithmic number of steps and again we seek to find a cut matching the achieved flow.

We discuss the need for different ϵ and ϵ' . In the bipartite case when a node decreased its weight, that node is guaranteed to receive no more allocation in the next round (it could remain the same though). Intuitively, this is important because in the above proof for bipartite graphs we want that if a node is in $\mathcal{G}(A)^+$, above the gap, if it was ever under-allocated then it never becomes over-allocated in later iterations. Our update ensures this will be the case since self-decreases will continue henceforth to keep the load of such a node below its capacity. Consider setting $\epsilon = \epsilon'$ for illustration. Because of the interaction between layers, a node i in B could receive more allocation even if it decreases its weight in an iteration. This is because the nodes in A and B could change their weights. Nodes in A changing their weight can give up to an extra $(1 + \epsilon)$ allocation (via predecessors of i that are

⁸ This averaging is what necessitates the poly-logarithmic number of weight update iterations in the algorithm

not decreased), and the same for B for a total of $(1 + \epsilon)^2$ extra allocation arriving at i . The node decreasing its weight reduces its allocation by a $(1 + \epsilon)$ factor for a total change of a $(1 + \epsilon)^2 \cdot \frac{1}{(1 + \epsilon)} > 1$ factor. By choosing ϵ and ϵ' to be different, as well as the characterization of which nodes decrease during a forced decrease, we can show any node will not receive less allocation if its weight does not decrease and will not receive more allocation if it performs a decrease. We call these properties “Increasing monotonicity” and “Decreasing monotonicity” in the full version of this paper.

As in the bipartite case, we can find a gap in layers A and B , which gives sets above the gap $\mathcal{G}(A)^+$ and $\mathcal{G}(B)^+$ in A and B , respectively. Let the sets $\mathcal{G}(A)^-$ and $\mathcal{G}(B)^-$ be the nodes below the gap. As before nodes in $\mathcal{G}(A)^-$ (resp., $\mathcal{G}(B)^-$) decreased their weight many more iterations than $\mathcal{G}(A)^+$ (resp., $\mathcal{G}(B)^+$). For simplicity, assume this partitions the nodes of the entire graph, so no nodes are inside either gap. Let $I_0 \subseteq I$ be the nodes adjacent to at least one node in $\mathcal{G}(A)^+$. Let A_0 be nodes in $\mathcal{G}(A)^-$, below the gap, that have an edge to $\mathcal{G}(B)^+$ (the analogue of I_0 in the A -layer). Any flow path that crosses the A layer at $\mathcal{G}(A)^+$, A_0 and $\mathcal{G}(A)^- \setminus A_0$ are blocked by the sets I_0, A_0 and $\mathcal{G}(B)^-$ respectively showing that this set forms a vertex cut.

We show the flow obtained is nearly the value of the vertex cut $I_0 \cup A_0 \cup \mathcal{G}(B)^-$. As before, nodes in I_0 (resp. A_0) send almost all their flow to nodes above the gap in the next layer $\mathcal{G}(A)^+$ (resp. $\mathcal{G}(B)^+$). Like before $\mathcal{G}(A)^+$ and $\mathcal{G}(B)^+$ do not decrease every round, so we can show they are not allocated more than their capacity (see Lemma 11 and Lemma 13 in this paper’s full version). The algorithmic key is that, by choosing the forced decrease carefully, we can show each node in $\mathcal{G}(A)^+$ has a neighbor in $\mathcal{G}(B)^+$. This ensures almost all of $\mathcal{G}(A)^+$ ’s flow reaches the sink because all of these nodes will send essentially all their the flow to $\mathcal{G}(B)^+$ and these nodes are not at capacity. Thus, I_0 can send all its flow to the sink successfully. Similarly, A_0 sends its flow to $\mathcal{G}(B)^+$ and then to the sink. Finally, as before, $\mathcal{G}(B)^-$ (as well as $\mathcal{G}(A)^-$) are sets of nodes near their allocation because they decreased essentially every iteration (see Lemma 12 and Lemma 14 in the full version). Thus, $\mathcal{G}(B)^-$ sends its flow directly to the sink. Moreover, we ensure that only a negligible fraction of this flow is double counted by the definition of the large weight reduction across the gaps (see Lemma 15 in the full version); therefore we have found a flow allocation obtained by the weights whose value nearly matches the value of an identified cut.

This analysis generalizes to layered DAGs where edges do not cross layers. To extend the existence of these weights to general DAGs we reduce the problem to finding weights with additional structural properties on a layered DAG. From the input DAG we make additional copies of nodes that have ancestors in earlier layers and link these copies via a path to the original neighbor. We convert any edge that crosses many layers to one in the layered DAG from its original head node to the copy of its tail node in the next layer. Then we argue that a key functional relation exists between the weights of any original node and its copies in earlier layers. This allows us to transfer the weights computed in the auxiliary layered DAG to the original DAG.

4 Matchings and Flows: Learnability of Predictions

We show that the weights are efficiently learnable. Assuming that each arriving impression is i.i.d. sampled from an unknown distribution, we want to learn a set of weights from a collection of past instances and examine their expected performance on a new instance

from the same distribution⁹. A direct approach might be to learn the unknown distribution from samples and utilize known ideas from stochastic optimization (where knowledge of the distribution is key). A major issue though is that there can be a large number of possible types (potentially exponential in the number of nodes of the DAG). A distribution that is sparse over types is not easy to learn with a small number of samples.

We claim that the weights are efficiently learnable, even if the distribution of types of impressions is not. We show that this task has low sample complexity and admits an efficient learning algorithm. Consequently, if there is an unknown arbitrary distribution that the impressions are drawn from, then only a small number of instances is required to compute the weights. The number of samples is proportional to size of the DAG without the impressions. In most problems such as Adwords, the number of arriving impressions is much larger than the fixed (offline) portion of the graph.

Before stating our results, we introduce two necessary assumptions. The first assumption is that each impression is i.i.d. sampled from an unknown distribution \mathcal{D} . Where no ambiguity will result, we also say an instance is sampled from \mathcal{D} if each impression is an i.i.d sample from \mathcal{D} . The second assumption is related to the expected instance of the distribution \mathcal{D} . The **expected instance** of a distribution is the instance where the number of each type of impressions is exactly the expected value.¹⁰ We assume that in the optimal solution of the expected instance, the load of each node is larger than a constant. Namely, it cannot happen that in the optimal flow, there exist many vertices which obtain very small amount of flow.

► **Theorem 4.** *Under the two assumptions above, for any $\epsilon, \delta \in (0, 1)$, there exists a learning algorithm such that, after observing $O(\frac{n^2}{\epsilon^2} \ln(\frac{n \log n}{\delta}))$ instances, returns weights $\{\hat{\alpha}\}$, satisfying that with probability at least $1 - \delta$, $\mathbb{E}_{I \sim \mathcal{D}}[R(\hat{\alpha}, I)] \geq (1 - \epsilon)\mathbb{E}_{I \sim \mathcal{D}}[R(\alpha^*, I)]$ where $R(\alpha, I)$ is the value of the fractional flow obtained by applying α to instance I and $\alpha^* = \arg \max_{\alpha} \mathbb{E}_{I \sim \mathcal{D}}[R(\alpha, I)]$.*

Technical Overview. Here we overview the analysis. The full proof is omitted in this version. To show that the weights are learnable we utilize a model similar to that of data-driven algorithm design. To illustrate our techniques we focus on the case when the instance is a bipartite graph $G = (I \cup A, E)$ with capacities C_a for each $a \in A$ (also recall that $|A| = n$).

In this setting there is an unknown distribution \mathcal{D} over instances of I of length m . The t 'th entry of I represents the t 'th impression arriving online. Our goal is to find a set of weights that performs well for the distribution \mathcal{D} . In particular let $\alpha^* := \arg \max_{\alpha \in \mathcal{S}} \mathbb{E}_{I \sim \mathcal{D}}[R(\alpha, I)]$ be the *best* set of weights for the distribution. Define $R(\alpha, I)$ to be the value of the matching using weights α on instance I . Here \mathcal{S} is a set of “admissible” weights, and in particular we only consider weights output by a proportional algorithm similar to the algorithm of Agrawal et al. [2]. We are allowed to sample s independent samples I_1, I_2, \dots, I_s from \mathcal{D} and use these samples to compute a set of weights $\hat{\alpha}$. We say that a learning algorithm (ϵ, δ) -learns the weights if with probability at least $1 - \delta$ over the samples from \mathcal{D} , we compute a set of weights $\hat{\alpha}$ satisfying $\mathbb{E}_{I \sim \mathcal{D}}[R(\hat{\alpha}, I)] \geq (1 - \epsilon)\mathbb{E}_{I \sim \mathcal{D}}[R(\alpha^*, I)]$.

This definition is similar to PAC learning [36]. Also note we are aiming for a relative error guarantee rather than an absolute error. The main quantity of interest is then the *sample complexity*, i.e. how large does s need to be as a function of n, m, ϵ , and δ in order to (ϵ, δ) -learn a set of weights? Ideally, s only depends polynomially on $n, m, 1/\epsilon$, and $1/\delta$, and smaller is always better.

⁹ We can also analyze the performance on similar distributions by applying techniques from our instance robustness result

¹⁰ Note this could be a fractional value.

The standard way to understand the sample complexity for this type of problem is via the *pseudo-dimension*. Intuitively, pseudo-dimension is the natural extension of VC-dimension to a class of real valued functions. In our case the class of functions is $\{R(\alpha, \cdot) \mid \alpha \in \mathcal{S}\}$, i.e. we are interested in the class of function mapping each instance I to the value of the fractional matching given by each fixed set of weights α . If the pseudo-dimension of this class of functions is d , then $s \approx \frac{d}{\epsilon^2} \log(1/\delta)$ samples are needed to (ϵ, δ) learn the weights, given that we are able to approximately optimize the empirical *average* performance [5].

The good news for our setting is that the pseudo-dimension of our class of functions is bounded. Each node in the set A can only have one of T different weight values for some parameter T . Then since the number of nodes in A is n , there can only be at most T^n different “admissible” weights. It is well known that the pseudo-dimension of a finite class of k different functions is $\log_2(k)$. Thus the pseudo-dimension of our class of functions is $d = n \log_2(T)$. As long as T isn’t growing too fast as a function of n and m , we get polynomial sample complexity.

Unfortunately, finding weights to optimize the average performance across the s sampled instances is complicated. Note that for a fixed instance I , the value of the matching as a function of the weights, $R(\cdot, I)$, is non-linear in the weights since we are using proportional allocation. Moreover, it is neither convex nor concave in the parameters α so applying a gradient descent approach will not work. Due to this, it is difficult to analyze the learnability via known results on pseudo-dimension.

The main tool we have at our disposal is that for a fixed instance I , we can compute weights α such that $R(\alpha, I) \geq (1 - \epsilon) \text{OPT}(I)$. This motivates the following natural direct approach. Take the s sampled instances I_1, I_2, \dots, I_s and take their union to form a larger “stacked” instance \hat{I} . We then run the aforementioned algorithm on \hat{I} to get weights $\hat{\alpha}$. Intuitively, if s is large enough, then by standard concentration inequalities $\hat{I} \approx s\mathbb{E}[I]$, i.e. the stacked instance approaches s copies of the “expected” instance. Then to complete the analysis, we need to show that $\mathbb{E}[R(\hat{\alpha}, I)] \approx R(\hat{\alpha}, \mathbb{E}[I])$. In general, it is not true that $\mathbb{E}[R(\hat{\alpha}, I)] \approx R(\hat{\alpha}, \mathbb{E}[I])$. Using more careful analysis, we show that when the distribution \mathcal{D} is a product distribution and our two assumptions hold, this is in fact the case.

5 Matching and Flows: Robustness

Instance Robustness. To show the instance robustness, we assume that we can describe the instance directly. Say we have a description of the entire instance denoted by a vector $\hat{I} = (\hat{m}_1, \dots, \hat{m}_i, \dots)$, where \hat{m}_i is the number of impressions of type i . We show that if a set of weights performs well in instance \hat{I} , it can be transferred to a nearby instance I robustly.

► **Theorem 5.** *For any $\epsilon > 0$, if a set of weights $\hat{\alpha}$ returns a $(1 - \epsilon)$ -approximated solution in instance \hat{I} , it yields an online flow allocation on instance I whose value is at least $\max\{(1 - \epsilon)\text{OPT} - 2\gamma, \text{OPT}/(d + 1)\}$. Here OPT is the maximum flow value on instance I , d is the diameter of this graph excluding vertex t , and γ is the difference between two instances, defined by $\|\hat{I} - I\|_1$.*

This theorem can be interpreted as follows. If we sample the instance and compute the weights in it, these weights will work well and break through worst-case bounds when the type proportions are sampled well. Indeed, the weights will perform well in nearby instances with similar type proportions. Moreover, the algorithm never performs worse than a $\frac{1}{d+1}$ factor of optimal. We remark that this is the best competitive ratio a deterministic integral algorithm can achieve because we show a lower bound on such algorithms (see Appendix G.2

in the full version of this paper). This example builds a recursive version of the simple lower bound of $\frac{1}{2}$ on the competitive ratio of deterministic online algorithms for the matching problem.

Recall that the key to showing existence of the weights was to construct a cut whose capacity is nearly the same as the value of the flow given by the weights. To show robustness against nearby instances, we observe how this proof can be extended to nearby cuts. This allows us to argue about the optimal value of the new instance. Standard calculations then let us connect the value of the predicted weights to this optimal value while losing only $O(\gamma)$ in the value of the flow. To ensure the algorithm is never worse than a $\frac{1}{d+1}$ factor of the optimal, we guarantee that the algorithm always returns a maximal allocation.

Parameter Robustness. For the parameter robustness, we show that the performance degrades linearly in the relative error of the weight parameter. Thus, our algorithm has the same robustness guarantees shown in other works as well.

► **Theorem 6.** *Consider a prediction of $\hat{\alpha}_v$ for each vertex $v \in V$. Due to scale invariance, we can assume that the minimum predicted vertex weight $\hat{\alpha}_{\min} = 1$. Define the prediction error $\eta := \max_{v \in V}(\frac{\hat{\alpha}_v}{\alpha_v^*}, \frac{\alpha_v^*}{\hat{\alpha}_v})$, where $\{\alpha_v^*\}_{v \in V}$ are vertex weights that can achieve an $(1 - \epsilon)$ -approximate solution and $\alpha_{\min}^* = 1$ for any fixed $\epsilon > 0$. Employing predictions $\hat{\alpha}$, we can obtain a solution with competitive ratio $\max(\frac{1}{d+1}, \frac{1-\epsilon}{\eta^{2d}})$, where d is the diameter of this graph excluding vertex t .*

When the prediction error η approaches one, the performance smoothly approaches optimal. While the above theorem involves comparing the propagation of the prediction errors in the performance analysis, we also investigate how inaccurate predictions can be adaptively corrected and improved in the 2-layered Adwords case. For that case, we give an improved algorithm that can correct error in the weights, so that the loss is only $O(\log \eta)$. Moreover, we show that the way we adapt is *tight* and the best possible up to constant factors for any algorithm given predicted weights with error η .

The parameter robustness follows almost directly from the definition of the proportional assignment given by the weights. In each layer, the over allocation (potentially above the capacity) can be easily bounded by a η^2 factor, resulting in a loss of at most η^{2d} on a d layer graph. We remark that directly using the weights ensures the algorithm is never worse than a $\frac{1}{d+1}$ factor of the optimal solution. The technical proof, along with the weight-adapting technique for the 2-layered case, is present in Appendix G of this paper's full version.

6 Results on Load Balancing

Next we show results for restricted assignment load balancing problem in our model. In particular, we study the instance robustness and learnability of proportional weights for this problem. The existence of useful weights and parameter robustness for predicting these weights were shown in prior work [2, 27]. As discussed before, we focus on analyzing fractional assignments.

To describe our results we introduce the following notation. Let $[m]$ denote the set of machines and S denote a set of jobs. Each job $j \in S$ has a size p_j and a neighborhood $N(j)$ of feasible machines. Given a set of positive weights $\{w_i\}_{i \in [m]}$ on the machines, we define a fractional assignment for each job j by setting $x_{ij}(w) = \frac{w_i}{\sum_{i' \in N(j)} w_{i'}}$ for each $i \in N(j)$. Let $\text{ALG}(S, w)$ be the fractional makespan on the jobs in S with weights w and similarly let $\text{OPT}(S)$ be the optimal makespan on the jobs in S . Prior work [2, 27] shows that for any set of jobs S and $\epsilon > 0$ there exists weights α such that $\text{ALG}(S, w) \leq (1 + \epsilon)\text{OPT}(S)$.

To describe our instance robustness result we consider an instance of the problem as follows. Consider instances with n jobs. The type of a job is the subset of machines to which it can be assigned. Let S_j be the total size of jobs of type j in instance S . We consider relative changes in the instance and define the difference between instances S and S' as $\eta(S, S') := \max_j \max\{\frac{S_j}{S'_j}, \frac{S'_j}{S_j}\}$. Our instance robustness result is given in the following theorem.

► **Theorem 7.** *For any instance S and $\epsilon > 0$, let w be weights such that $\text{ALG}(S, w) \leq (1 + \epsilon)\text{OPT}(S)$. Then for any instance S' we have $\text{ALG}(S', w) \leq (1 + \epsilon)^2 \eta(S, S')^2 \text{OPT}(S')$.*

Let w be as in the statement of the theorem and w' be weights such that $\text{ALG}(S', w') \leq (1 + \epsilon)\text{OPT}(S')$. Intuitively, we lose the first factor of $\eta(S, S')$ by bounding the performance of w on S' and the second factor by bounding the performance of w' on S .

Next we study learnability. We give the first result showing these weights are learnable in any model. In order to understand the sample complexity of learning the weights we need to consider an appropriate discretization of the space of possible weights. For integer $R > 0$ and $\epsilon > 0$, let $\mathcal{W}(R) = \{\alpha \in \mathbb{R}^m \mid \alpha_i = (1 + \epsilon)^k, i \in [m], k \in \{0, 1, \dots, R\}\}$. Additionally, let p_{\max} be an upper bound on all jobs sizes. The following theorem characterizes the learnability of the weights for restricted assignment load balancing.

► **Theorem 8.** *Let $\epsilon, \delta \in (0, 1)$ be given and set $R = O(\frac{m^2}{\epsilon^2} \log(\frac{m}{\epsilon}))$ and let $\mathcal{D} = \prod_{j=1}^n \mathcal{D}_j$ be a product distribution over n -job restricted assignment instances such that $\mathbb{E}_{S \sim \mathcal{D}}[\text{OPT}(S)] \geq \Omega(\frac{1}{\epsilon^2} \log(\frac{m}{\epsilon}))$. There exists an algorithm which finds weights $w \in \mathcal{W}(R)$ such that $\mathbb{E}_{S \sim \mathcal{D}}[\text{ALG}(S, w)] \leq (1 + \epsilon)\mathbb{E}_{S \sim \mathcal{D}}[\text{OPT}(S)]$ with probability at least $1 - \delta$ when given access to $s = \tilde{O}(\frac{m^3}{\epsilon^2} \log(\frac{1}{\delta}))$ independent samples $S_1, S_2, \dots, S_s \sim \mathcal{D}$.*

Our techniques here are similar to that of the online flow allocation problem in that we use the samples to construct a “stacked” instance then compute a set of near optimal weights on this instance. We then have to show that these weights work well in expectation with high probability. This step necessitates the two assumptions in the theorem statement. First, we need that the expected optimal makespan is reasonably large so that the expected makespan is close to the maximum of the expected loads of the machines. Second, we need that the instance is drawn from a product distribution so that the stacked instance converges in some sense to s copies of the “expected” instance. See the full version of the paper for complete arguments.

References

- 1 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014. doi:10.1287/opre.2014.1289.
- 2 Shipra Agrawal, Morteza Zadimoghaddam, and Vahab Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL: <http://proceedings.mlr.press/v80/agrawal18b.html>.
- 3 Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms. *ICML 2020*, 2020.
- 4 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle*,

- Washington, USA, volume 151 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.52.
- 5 Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, USA, 1st edition, 2009.
 - 6 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *CoRR*, abs/2003.02144, 2020. arXiv:2003.02144.
 - 7 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *CoRR*, abs/2006.01026, 2020. arXiv:2006.01026.
 - 8 Yossi Azar, Joseph Seffi Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995. doi:10.1006/jagm.1995.1008.
 - 9 Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? *CoRR*, abs/1908.02894, 2019. arXiv:1908.02894.
 - 10 Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 353–362. PMLR, 2018. URL: <http://proceedings.mlr.press/v80/balcan18a.html>.
 - 11 Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 603–614. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00064.
 - 12 Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized lloyd’s families. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10664–10674, 2018. URL: <http://papers.nips.cc/paper/8263-data-driven-clustering-via-parameterized-lloyds-families>.
 - 13 Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. *CoRR*, abs/2002.04726, 2020. arXiv:2002.04726.
 - 14 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, 2016. doi:10.1145/2993749.2993766.
 - 15 Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Pandora’s box with correlations: Learning and approximation, 2020. arXiv:1911.01632.
 - 16 Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78, 2009.
 - 17 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013. doi:10.1137/1.9781611973105.7.
 - 18 Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010. doi:10.1007/978-3-642-15775-2_16.

- 19 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019. URL: <http://proceedings.mlr.press/v97/gollapudi19a.html>.
- 20 Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Math. Oper. Res.*, 41(4):1404–1431, 2016. doi:10.1287/moor.2016.0782.
- 21 Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, 46(3):992–1017, 2017. doi:10.1137/15M1050276.
- 22 Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrovic, and Ronitt Rubinfeld. Online page migration with ML advice. *CoRR*, abs/2006.05028, 2020. arXiv:2006.05028.
- 23 Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.69.
- 24 Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000. doi:10.1016/S0304-3975(99)00140-1.
- 25 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990. doi:10.1145/100216.100262.
- 26 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18*, pages 489–504, New York, NY, USA, 2018. ACM.
- 27 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. doi:10.1137/1.9781611975994.114.
- 28 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL: <http://proceedings.mlr.press/v80/lykouris18a.html>.
- 29 Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Trans. Algorithms*, 8(1):2:1–2:29, 2012. doi:10.1145/2071379.2071381.
- 30 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007. doi:10.1145/1284320.1284321.
- 31 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 462–471, 2018.
- 32 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions, 2020. arXiv:2006.09123.
- 33 Marco Molinaro and R. Ravi. Geometry of online packing linear programs. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 701–713. Springer, 2012. doi:10.1007/978-3-642-31594-7_59.

- 34 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 9684–9693, 2018. URL: <http://papers.nips.cc/paper/8174-improving-online-algorithms-via-ml-predictions>.
- 35 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020. doi:10.1137/1.9781611975994.112.
- 36 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.
- 37 Alexander Wei. Better and simpler learning-augmented online caching. *CoRR*, abs/2005.13716, 2020. [arXiv:2005.13716](https://arxiv.org/abs/2005.13716).
- 38 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling, 2020. [arXiv:2010.11629](https://arxiv.org/abs/2010.11629).
- 39 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms, 2020. [arXiv:2010.11632](https://arxiv.org/abs/2010.11632).

Telescoping Filter: A Practical Adaptive Filter

David J. Lee ✉

Cornell University, Ithaca, NY, USA

Samuel McCauley ✉🏠

Williams College, Williamstown, MA, USA

Shikha Singh ✉🏠

Williams College, Williamstown, MA, USA

Max Stein ✉

Williams College, Williamstown, MA, USA

Abstract

Filters are small, fast, and approximate set membership data structures. They are often used to filter out expensive accesses to a remote set \mathcal{S} for negative queries (that is, filtering out queries $x \notin \mathcal{S}$). Filters have one-sided errors: on a negative query, a filter may say “present” with a tunable false-positive probability of ε . Correctness is traded for space: filters only use $\log(1/\varepsilon) + O(1)$ bits per element.

The false-positive guarantees of most filters, however, hold only for a single query. In particular, if x is a false positive, a subsequent query to x is a false positive with probability 1, not ε . With this in mind, recent work has introduced the notion of an **adaptive filter**. A filter is adaptive if each query is a false positive with probability ε , regardless of answers to previous queries. This requires “fixing” false positives as they occur.

Adaptive filters not only provide strong false positive guarantees in adversarial environments but also improve query performance on practical workloads by eliminating repeated false positives.

Existing work on adaptive filters falls into two categories. On the one hand, there are practical filters, based on the cuckoo filter, that attempt to fix false positives heuristically without meeting the adaptivity guarantee. On the other hand, the broom filter is a very complex adaptive filter that meets the optimal theoretical bounds.

In this paper, we bridge this gap by designing the **telescoping adaptive filter** (TAF), a practical, provably adaptive filter. We provide theoretical false-positive and space guarantees for our filter, along with empirical results where we compare its performance against state-of-the-art filters. We also implement the broom filter and compare it to the TAF. Our experiments show that theoretical adaptivity can lead to improved false-positive performance on practical inputs, and can be achieved while maintaining throughput that is similar to non-adaptive filters.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Filters, approximate-membership query data structures (AMQs), Bloom filters, quotient filters, cuckoo filters, adaptivity, succinct data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.60

Related Version *Full Version*: <https://arxiv.org/abs/2107.02866>

Supplementary Material *Software*: <https://github.com/djslzx/telescoping-filter>
archived at `swb:1:dir:76164f338679e3e058408d6a5572d400e2f03977`

Funding *David J. Lee*: This author’s research is supported in part by NSF CCF 1947789.

Samuel McCauley: This author’s research is supported in part by NSF CCF 2103813

Shikha Singh: This author’s research is supported in part by NSF CCF 1947789.

Max Stein: This author’s research is supported in part by NSF CCF 1947789.



© David J. Lee, Samuel McCauley, Shikha Singh, and Max Stein;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 60; pp. 60:1–60:18

Leibniz International Proceedings in Informatics



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

1 Introduction

A filter is a compact and probabilistic representation of a set \mathcal{S} from a universe \mathcal{U} . A filter supports insert and query operations on \mathcal{S} . On a query for an element $x \in \mathcal{S}$, a filter returns “present” with probability 1, i.e., a filter guarantees no false negatives. A filter is allowed to have bounded false positives – on a query for an element $x \notin \mathcal{S}$, it may incorrectly return “present” with a small and tunable false-positive probability ε .

Filters are used because they allow us to trade correctness for space. A lossless representation of $\mathcal{S} \subseteq \mathcal{U}$ requires $\Omega(n \log u)$ bits, where $n = |\mathcal{S}|$, $u = |\mathcal{U}|$, and $n \ll u$. Meanwhile, an optimal filter with false-positive probability ε requires only $\Theta(n \log(1/\varepsilon))$ bits [11].

Examples of classic filters are the *Bloom filter* [7], the *cuckoo filter* [18], and the *quotient filter* [6]. Recently, filters have exploded in popularity due to their widespread applicability – many practical variants of these classic filters have been designed to improve upon throughput, space efficiency, or cache efficiency [8, 16, 19, 22, 29, 32].

A filter’s small size allows it to fit in fast memory, higher in the memory hierarchy than a lossless representation of \mathcal{S} would allow. For this reason, filters are frequently used to speed up expensive queries to an external dictionary storing \mathcal{S} .

In particular, when a dictionary for \mathcal{S} is stored remotely (on a disk or across a network), checking a small and fast filter first can avoid expensive remote accesses for a $1 - \varepsilon$ fraction of negative queries. This is the most common use case of the filter, with applications in LSM-based key-value stores [12, 23, 27], databases [13, 15, 17], and distributed systems and networks [9, 31].

False positive guarantees and adaptivity. When a filter is used to speed up queries to a remote set \mathcal{S} , its performance depends on its false-positive guarantees: how often does the filter make a mistake, causing us to access \mathcal{S} unnecessarily?

Many existing filters, such as the Bloom, quotient and cuckoo filters, provide poor false-positive guarantees because they hold only for a single query. Because these filters do not *adapt*, that is, they do not “fix” any false positives, querying a known false positive x repeatedly can drive their false-positive rate to 1, rendering the filter useless.

Ideally, we would like a stronger guarantee: even if a query x has been a false positive in the past, a subsequent query to x is a false positive with probability at most ε . This means that the filter must “fix” each false positive x as it occurs, so that subsequent queries to x are unlikely to be false positives. This notion of adaptivity was formalized by Bender et al. [5]. A filter is **adaptive** if it guarantees a false positive probability of ε for every query, *regardless of answers to previous queries*. Thus, adaptivity provides security advantages against an adversary attempting to degrade performance, e.g., in denial-of-service attacks.

At the same time, fixing previous false positives leads to improved performance. Many practical datasets do, in fact, repeatedly query the same element – on such a dataset, fixing previous false positives means that a filter only incurs one false positive per *unique* query. Past work has shown that simple, easy-to-implement changes to known filters can fix false positives heuristically. Due to repeated queries, these heuristic fixes can lead to reduction of several orders of magnitude in the number of incurred false positives [10, 21, 25].

Recent efforts that tailor filters to query workloads by applying machine learning techniques to optimize performance [14, 24, 30] reinforce the benefits achieved by adaptivity.

Adaptivity vs practicality. The existing work on adaptivity represents a dichotomy between simple filters one would want to implement and use in practice but are not actually adaptive [21, 25], or adaptive filters that are purely theoretical and pose a challenge to implementation [5].

Mitzenmacher et al. [25] provided several variants of the *adaptive cuckoo filter* (ACF) and showed that they incurred significantly fewer false positives (compared to a standard cuckoo filter) on real network trace data. The data structures in [25] use simple heuristics to fix false positives with immense practical gains, leaving open the question of whether such heuristics can achieve worst-case guarantees on adaptivity.

Recently, Kopelowitz et al. [21] proved that this is not true even for a non-adversarial notion of adaptivity. In particular, they defined **support optimality** as the adaptivity guarantee on “predetermined” query workloads: that is, query workloads that are fixed ahead of time and not constructed in response to a filter’s response on previous queries. They showed that the filters in [25] fail to be adaptive even under this weaker notion – repeating $O(1)$ queries n times may cause them to incur $\Omega(n)$ false positives.

Kopelowitz et al. [21] proposed a simple alternative, the *cuckooing ACF*, that achieves support optimality by cuckooing on false positives (essentially reinserting the element). Furthermore, they proved that none of the cuckoo filter variants (including the cuckooing ACF) are adaptive. They showed that a prerequisite to achieving adaptivity is allocating a variable number of bits to each stored element – that is, maintaining variable-length fingerprints. All of the cuckooing filter variants use a bounded number of bits per element.

The only known adaptive filter is the **broom filter** of Bender et al. [5], so-named because it “cleans up” its mistakes. The broom filter achieves adaptivity, while supporting constant-time worst-case query and insert costs, using very little extra space – $O(n)$ extra bits in total. Thus the broom filter implies that, *in theory*, adaptivity is essentially free.

More recently, Bender et al. [4] compared the broom filter [5] to a static filter augmented with a comparably sized top- k cache (a cache that stores the k most frequent requests). They found that the broom filter outperforms the cache-augmented filter on Zipfian distributions due to “serendipitous corrections” – fixing a false positive eliminates future false positives in addition to the false positive that triggered the adapt operation. They noted that their broom filter simulation is “quite slow,” and left open the problem of designing a practical broom filter with performance comparable to that of a quotient filter.

In this paper, we present a practical and efficient filter which also achieves worst-case adaptivity: the **telescoping adaptive filter**. The key contribution of this data structure is a practical method to achieve worst-case adaptivity using variable-length fingerprints.

Telescoping adaptive filter. The telescoping adaptive filter (TAF) combines ideas from the heuristics used in the adaptive cuckoo filter [25], and the theoretical adaptivity of the broom filter [5].

The TAF is built on a rank-and-select quotient filter (RSQF) [29] (a space- and cache-efficient quotient filter [6] variant), and inherits its performance guarantees.

The telescoping adaptive filter is the first adaptive filter that can take advantage of *any* amount of extra space for adaptivity, even a fractional number of bits per element. We prove that if the TAF uses $\left(\frac{1}{\epsilon} + \frac{b}{(1-b)^2}\right)$ extra bits per element in expectation, then it is provably adaptive for any workload consisting of up to $n/(b\sqrt{\epsilon})$ unique queries (Section 4). Empirically, we show that the TAF outperforms this bound: with only 0.875 of a bit extra per element for adaptivity, it is adaptive for larger query workloads. Since the RSQF uses 2.125 metadata bits per element, the total number of bits used by the TAF is $(n/\alpha)(\log_2(1/\epsilon) + 3)$, where α is the load factor.

The TAF stores these extra bits space- and cache-efficiently using a practical implementation of a theoretically-optimal compression scheme: *arithmetic coding* [20,33]. Arithmetic coding is particularly well-suited to the exponentially decaying probability distribution of repeated false positives. While standard arithmetic coding on the unit interval can be slow, we implement an efficient approximate integer variant.

The C code for our implementation can be found at <https://github.com/djslzx/telescoping-filter>.

Our contributions. We summarize our main contributions below.

- We present the first provably-adaptive filter, the telescoping adaptive filter, engineered with space, cache-efficiency and throughput in mind, demonstrating that adaptivity is not just a theoretical concept, and can be achieved in practice.
- As a benchmark for TAF, we also provide a practical implementation of the broom filter [5]. We call our implementation of the broom filter the **extension adaptive filter** (exAF). While both TAF and exAF use the near-optimal $\Theta(n)$ extra bits in total to adapt on $\Theta(n)$ queries, the telescoping adaptive filter is optimized to achieve better constants and eke out the most *adaptivity per bit*. This is confirmed by our experiments which show that given the same space for adaptivity (0.875 bits per element), the TAF outperforms the false-positive performance of the exAF significantly on both practical and adversarial workloads. Meanwhile, our experiments show that the query performance of exAF is factor 2 better than that of the TAF. Thus, we show that there is a trade off between throughput performance and how much adaptivity is gained from each bit.
- We give the first empirical evaluation of how well an adaptive filter can fix positives in practice. We compare the TAF with a broom filter implementation, as well as with previous heuristics. We show that the TAF frequently matches or outperforms other filters, while it is especially effective in fixing false positives on “difficult” datasets, where repeated queries are spaced apart by many other false positives. We also evaluate the throughput of the TAF and the exAF against the vacuum filter [32] and RSQF, showing for the first time that adaptivity can be achieved while retaining good throughput bounds.

2 Preliminaries

In this section, we provide background on filters and adaptivity, and describe our model.

2.1 Background on Filters

We briefly summarize the structure of the filters discussed in this paper. For a more detailed description, we refer the reader to the full version. All logs in the paper are base 2. We assume that ε is an inverse power of 2.

The quotient filter and cuckoo filter are both based on the single-hash function filter [28]. Let the underlying hash function h output $\Theta(\log n)$ bits. To represent a set $\mathcal{S} \subseteq \mathcal{U}$, the filter stores a fingerprint $f(x)$ for each element $x \in \mathcal{S}$. The fingerprint $f(x)$ consists of the first $\log n + \log(1/\varepsilon)$ bits of $h(x)$, where $n = |\mathcal{S}|$ and ε is the false-positive probability.

The first $\log n$ bits of $f(x)$ are called the **quotient** $q(x)$ and are stored implicitly; the remaining $\log(1/\varepsilon)$ bits are called the **remainder** $r(x)$ and are stored explicitly in the data structure. Both filters consist of an array of slots, where each slot can store one remainder.

Quotient filter. The quotient filter (QF) [6] is based on linear probing. To insert $x \in \mathcal{S}$, the remainder $r(x)$ is stored in the slot location determined by the quotient $q(x)$, using linear probing to find the next empty slot. A small number of metadata bits suffice to recover the original slot for each stored element. A query for x checks if the remainder $r(x)$ is stored in the filter – if the remainder is found, it returns “present”; otherwise, it returns “absent.” The rank-and-select quotient filter (RSQF) [29] implements such a scheme using very few metadata bits (only 2.125 bits) per element.

Broom filter. The **broom filter** of Bender et al. [5] is based on the quotient filter. Initially, it stores the same fingerprint $f(x)$ as a quotient filter. The broom filter uses the remaining bits of $h(x)$, called **adaptivity bits**, to extend $f(x)$ in order to adapt on false positives.

On a query y , if there is an element $x \in \mathcal{S}$ such that $f(x)$ is a prefix of $h(y)$, the broom filter returns “present.” If it turns out that $y \notin \mathcal{S}$, the broom filter adapts by extending the fingerprint $f(x)$ until it is no longer a prefix of $h(y)$.¹ Bender et al. show that, with high probability, $O(n)$ total adaptivity bits of space are sufficient for the broom filter to be adaptive on $\Theta(n)$ queries.

Cuckoo filters and adaptivity. The cuckoo filter resembles the quotient filter but uses a cuckoo hash table rather than linear probing. Each element has two fingerprints, and therefore two quotients. The remainder of each $x \in \mathcal{S}$ must always be stored in the slot corresponding to one of x ’s two quotients.

The Cyclic ACF [25], Swapping ACF [25], and Cuckooing ACF [21]² change the function used to generate the remainder on a false positive. To avoid introducing false negatives, a filter using this technique must somehow track which function was used to generate each remainder so that the appropriate remainders can be compared at query time.

The Cyclic ACF stores s extra bits for each slot, denoting which of 2^s different remainders are used. The Swapping ACF, on the other hand, groups slots into constant-sized bins, and has a fixed remainder function for each slot in a bin. A false positive is fixed by moving some $x \in \mathcal{S}$ to a different slot in the bin, then updating its remainder using the function corresponding to the new slot. The Cuckooing ACF works in much the same way, but both the quotient and remainder are changed by “cuckooing” the element to its alternate position in the cuckoo table.

2.2 Model and Adaptivity

All filters that adapt on false positives [5, 21, 25] have access to the original set \mathcal{S} . This is called the *remote representation*, denoted \mathbf{R} . The remote representation does not count towards the space usage of the filter. On a false positive, the filter is allowed to access the set \mathbf{R} to help fix the false positive.

The justification for this model is twofold. (This justification is also discussed in [5, 21, 25].) First, the most common use case of filters is to filter out negative queries to \mathcal{S} – in this case, a positive response to a query accesses \mathbf{R} anyway. Information to help rebuild the filter can be stored alongside the set in this remote database. Second, remote access is necessary to achieve good space bounds: Bender et al. [5] proved that any adaptive filter without remote access to \mathcal{S} requires $\Omega(n \log \log u)$ bits of space.

Our filter can answer queries using only the local state \mathbf{L} . Our filter accesses the remote state \mathbf{R} in order to fix false positives when they occur, updating its local state. This allows our filter to be adaptive while using small (near optimal) space for the local state.

Adaptivity. The **sustained false positive** rate of a filter is the probability with which a query is a false positive, *regardless of the filter’s answers to previous queries*.

¹ These additional bits are stored separately in the broom filter: groups of $\Theta(\log n)$ adaptivity bits, corresponding to $\log n$ consecutive slots in the filter, are stored such that accessing all the adaptivity bits of a particular element (during a query operation) can be done in $O(1)$ time.

² We use the nomenclature of [21] in calling these the Cyclic ACF, Swapping ACF, and Cuckooing ACF.

The sustained false positive rate must hold even if generated by an adversary. We use the definition of Bender et al. [5], which is formally defined by a game between an adversary and the filter, where the adversary’s goal is to maximize the filter’s false positive rate. We summarize this game next; for a formal description of the model see Bender et al. [5].

In the **adaptivity game**, the adversary generates a sequence of queries $Q = x_1, x_2, \dots, x_t$. After each query x_i , both the adversary and filter learn whether x_i is a false positive (that is, $x_i \notin S$ but a query on x_i returns “present”). The filter is then allowed to adapt before query x_{i+1} is made by the adversary. The adversary can use the information about whether queries x_1, \dots, x_i were a false positive or not, to choose the next query x_{i+1} .

At any time t , the adversary may assert that it has discovered a special query \tilde{x}_t that is likely to be a false positive of the filter. The adversary “wins” if \tilde{x}_t is in fact a false positive of the filter at time t , and the filter “wins” if the adversary is wrong and \tilde{x}_t is not a false positive of the filter at time t .

The **sustained false positive** rate of a filter is the maximum probability ε with which the adversary can win the above adaptivity game. A filter is **adaptive** if it can achieve a sustained false positive rate of ε , for any constant $0 < \varepsilon < 1$.

Similar to [5], we assume that the adversary cannot find a never-before-queried element that is a false positive of the filter with probability greater than ε . Many hash functions satisfy this property, e.g., if the adversary is a polynomial-time algorithm then one-way hash functions are sufficient [26]. Cryptographic hash functions satisfy this property in practice, and it is likely that even simple hash functions (like Murmurhash used in this paper) suffice for most applications.

Towards an adaptive implementation. Kopelowitz et al. [21] showed that the Cyclic ACF (with any constant number of hash-selector bits), the Swapping ACF, and the Cuckooing ACF are not adaptive. The key insight behind this proof is that for all three filters, the state of an element – which slot it is stored in, and which fingerprint function is used – can only have $O(1)$ values. Over $o(n)$ queries, an adversary can find queries that collide with an element on all of these states. These queries can never be fixed.

Meanwhile, the broom filter avoids this issue by allowing certain elements to have more than $O(1)$ adaptivity bits – up to $O(\log n)$, in fact. The broom filter stays space-efficient by maintaining $O(1)$ adaptivity bits per element *on average*.

Thus, a crucial step for achieving adaptivity is dynamically changing how much space is used for the adaptivity of each element based on past queries. The telescoping adaptive filter achieves this dynamic space allocation (hence the name “telescoping”) using an arithmetic coding.

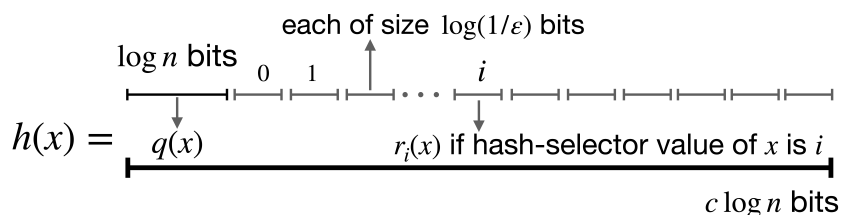
3 The Telescoping Adaptive Filter

In this section, we describe the high-level ideas behind the telescoping adaptive filter.

Structure of the telescoping adaptive filter. Like the broom filter, the TAF is based on a quotient filter where the underlying hash function h outputs $\Theta(\log n)$ bits. For any $x \in S$, the first $\log n$ bits of $h(x)$ are the quotient $q(x)$ (stored implicitly), and the next $\log(1/\varepsilon)$ bits are the initial remainder $r_0(x)$, stored in the slot determined by the quotient. We maintain each element’s original slot using the strategy of the rank-and-select quotient filter [29], which stores 2.125 metadata bits per element.

The TAF differs from a broom filter in that, on a false positive, the TAF *changes* its remainder rather than lengthening it, similar to the Cyclic ACF.

For each element in the TAF, we store a **hash-selector value**. If an element x has hash-selector value i , its remainder r_i is the consecutive sequence of $\log(1/\varepsilon)$ bits starting at the $(\log n + i \log(1/\varepsilon))$ th bit of $h(x)$. Initially, the hash-selector values of all elements are 0, and thus the remainder $r(x)$ is the first $\log 1/\varepsilon$ bits immediately following the quotient. When the hash-selector value of an element $x \in S$ is incremented, its remainder “slides over” to the next (non-overlapping) $\log 1/\varepsilon$ bits of the hash $h(x)$, as shown in Figure 1. Thus, the fingerprint of x is $f(x) = q(x) \circ r_i(x)$, where \circ denotes concatenation and i is the hash-selector value of x .



■ **Figure 1** The fingerprint of $x \in S$ is its quotient $q(x)$ followed by its remainder $r_i(x)$, where i is the hash-selector value of x .

On a false positive query $y \notin S$, there must be some $x \in S$ with hash-selector value i , such that $q(x) = q(y)$ and $r_i(x) = r_i(y)$. To resolve this false positive, we increment i . We update the hash-selector value and the stored remainder accordingly.

We describe below how to store hash-selector values using an average 0.875 bits per element. This means that the TAF with load factor α uses $(n/\alpha)(\log(1/\varepsilon) + 3)$ bits of space.

Difference between hash-selector and adaptivity bits. Using hash-selector bits, rather than adaptivity bits (as in the broom filter), has some immediate upsides and downsides.

If fingerprint prefixes $p(x)$ and $p(y)$ collide, they will still collide with probability $1/2$ after each prefix has been lengthened by one bit. But adding a bit also reduces the probability that x will collide with any *future* queries by a factor of $1/2$. Such false positives that are fixed (without being queried) are called **serendipitous false positives** [4].

On the other hand, incrementing the hash-selector value of an element $x \in S$ after it collides with an element $y \notin S$ reduces the probability that y will collide again with x by a factor of $\varepsilon \ll 1/2$. Thus, the TAF is more aggressive about fixing repeated false positives. However, the probability that x collides with future queries that are different from y remains unchanged. Thus, on average the TAF does not fix serendipitous false positives.

Our experiments (Section 6) show that the gain of serendipitous false positive fixes is short-lived; aggressively fixing false positives leads to better false-positive performance.

Storing hash selectors in blocks. The TAF does not have a constant number of bits per slot dedicated solely to storing its hash-selector value. Instead, we group the hash-selector values associated with each $\Theta(\log n)$ contiguous slots (64 slots in our implementation) together in a block. We allocate a constant amount of space for each such block. If we run out of space,

we **rebuild** by setting all hash-selector values in the block to 0. (After a rebuild, we still fix the false positive that caused the rebuild. Therefore, there will often be one non-zero hash-selector value in the block after a rebuild.)

Encoding hash-selector bits. To store the hash selectors effectively, we need a code that satisfies the following requirements: the space of the code should be very close to optimal; the code should be able to use < 1 bits on average per character encoded; and the encode and decode operations should be fast enough to be usable in practice.

In Section 5, we give a new implementation of the arithmetic coding that is tailored to our use case, specifically encoding characters from the distribution given in Corollary 3. Our implementation uses only integers, and all divisions are implemented using bit shifts, leading to a fast and reliable implementation while still retaining good space bounds.

4 Telescoping Adaptive Filter: Analysis

In this section, we analyze the sustained false-positive rate, the hash-selector probabilities, and the space complexity of the telescoping adaptive filter.

We assume the TAF uses a uniform random hash function h such that the hash can be evaluated in $O(1)$ time. In our adaptivity analysis of the TAF (Theorem 1), we first assume that the filter has sufficient space to store all hash-selector values; that is, it does not rebuild. Then, in Theorem 4, we give a bound on the number of unique queries that the TAF can handle (based on its size) without the need to rebuild, thus maintaining adaptivity.

Adaptivity. We first prove that the telescoping adaptive filter is adaptive, i.e., it guarantees a sustained false positive rate of ε .

We say a query x has a **soft collision** with an element $y \in \mathcal{S}$ if their quotients are the same: $q(x) = q(y)$. We say a query x has a **hard collision** with an element $y \in \mathcal{S}$ if both their quotients and remainders are the same: $q(x) = q(y)$ and $r_i(x) = r_i(y)$, where i is the hash-selector value of y at the time x is queried (see Section 3).

► **Theorem 1.** *Consider a telescoping adaptive filter storing a set \mathcal{S} of size n . For any adaptively generated sequence of t queries $Q = x_1, x_2, \dots, x_t$ (possibly interleaved with insertions), where each $x_i \notin \mathcal{S}$, the TAF has a sustained false-positive rate of ε ; that is, $\Pr[x_i \text{ is a false positive}] \leq \varepsilon$ for all $1 \leq i \leq t$.*

Proof. Consider the i -th query $x_i \in Q$. Query x_i is a false positive if there exists an element $y \in \mathcal{S}$ such that there is hard collision between them. Let $h_i(y) = q(y) \circ r_k(y)$ denote the fingerprint of y at time i , where y has the hash-selector value k at time i . Then, x_i and y have a hard collision if and only if $h_i(x_i) = h_i(y)$.

We show that for any y , regardless of answers to previous queries, x_i and y have a hard collision with probability ε/n ; taking a union bound over all elements gives the theorem.

We proceed in cases. First, if x_i is a first-time query, that is, $x_i \notin \{x_1, \dots, x_{i-1}\}$, then the probability that $h_i(x_i) = h_i(y)$ is the probability that both their quotient and remainder match, which occurs with probability $2^{-(\log n + \log 1/\varepsilon)} = \varepsilon/n$.

Next, suppose that x_i is a repeated query, that is, $x_i \in \{x_1, \dots, x_{i-1}\}$. Let $j < i$ be the largest index where $x_i = x_j$ was previously queried. If x_j did not have a soft collision with y , that is, $q(x_j) \neq q(y)$, then x_i cannot have a hard collision with y . Now suppose that $q(x_j) = q(y)$. We have two subcases.

1. y 's hash-selector value has not changed since x_j was queried. Note that, in this case, x_j must not have had a hard collision with y , as that would have caused y 's hash-selector value, and thus its remainder, to be updated. Thus, $h_j(y) = h_i(y) \neq h_j(x_j) = h_i(x_i)$.
2. y 's hash-selector value has been updated since x_j was queried. Such an update could have been caused by a further query to x_j having a hard collision with y , or some other query $x_k \in x_j, x_{j+1}, \dots, x_i$ having a hard collision with y . In either case, the probability that the new remainder matches, i.e., $r_i(y) = r_i(x_i)$, is $2^{-\log 1/\varepsilon} = \varepsilon$.

Therefore, the probability that x_i has a hard collision with y is at most $\varepsilon \cdot \Pr[q(x_j) = q(y)] = \varepsilon/n$. Finally, by a union bound over n possibilities for $y \in \mathcal{S}$, we obtain that $\Pr[x_i \text{ is a false positive}] \leq \varepsilon$ for all $1 \leq i \leq t$, as desired. \blacktriangleleft

Hash-selector probabilities. The telescoping adaptive filter increments the hash-selector value of an element $y \in \mathcal{S}$ whenever a false positive query collides with y . Here we analyze the probability of an element having a given hash-selector value.

► **Lemma 2.** *Consider a sequence $Q = x_1, x_2, \dots, x_t$ of queries (interleaved with inserts), where each $x_i \notin \mathcal{S}$ and Q consists of cn unique queries (with any number of repetitions), where $c < 1/\varepsilon - 1$. Then for any $y \in \mathcal{S}$, if $v(y)$ is the hash-selector value of y after all queries in Q are performed, then:*

$$\Pr[v(y) = k] \begin{cases} = (1 - \frac{\varepsilon}{n})^{cn} & \text{if } k = 0 \\ \leq \varepsilon^k (1 - \varepsilon) \sum_{i=1}^k \binom{cn}{i} \frac{1}{n^i} & \text{if } k > 0 \end{cases}$$

Proof. First, consider the case $k = 0$: the hash-selector value of y stays zero after all the queries are made if and only if none of the queries have a hard collision with y . Since there are cn unique queries, and the probability that each of them has a hard collision with y is ε/n , the probability that none of them collide with y is $(1 - \varepsilon/n)^{cn}$.

Now, consider the case $k \geq 1$. Given that the hash selector value of y is k , we know that there have been exactly k hard collisions between queries and y (where some of these collisions may have been caused by the same query). Suppose there are i unique queries among all queries that have a hard collision with y , where $1 \leq i \leq k$. Let k_j be the number of times a query j collides with y causing an increment in its hash-selector value, where $1 \leq j \leq i$. Thus, $\sum_{j=1}^i k_j = k$.

For a query x_j , the probability that x_j collides with y , the first time x_j is queried, is ε/n . Then, given that x_j has collided with y once, the probability of any subsequent collision with y is ε . (This is because the $\log 1/\varepsilon$ bits of the remainder of y are updated with each collision.) Thus, the probability that x_j collides with y at least k_j times is $\frac{\varepsilon}{n} \cdot \varepsilon^{k_j-1}$.

The probability that a query x_j collides with y at least k_j times, is given by $\prod_{j=1}^i \frac{\varepsilon}{n} \cdot \varepsilon^{k_j-1} = \frac{\varepsilon^k}{n^i}$. There are $\binom{cn}{i}$ ways of choosing i unique queries from cn , for $1 \leq i \leq k$, which gives us

$$\Pr[v(y) \geq k] = \varepsilon^k \sum_{i=1}^k \binom{cn}{i} \frac{1}{n^i} \tag{1}$$

Finally, using Inequality 1, we can upper bound the probability that a hash-selector value is exactly k .

$$\begin{aligned}
\Pr[v(y) = k] &= \Pr[v(y) \geq k] - \Pr[v(y) \geq k+1] \\
&= \varepsilon^k \left[\sum_{i=1}^k \binom{cn}{i} \frac{1}{n^i} - \varepsilon \sum_{i=1}^{k+1} \binom{cn}{i} \frac{1}{n^i} \right] \\
&= \varepsilon^k \cdot (1 - \varepsilon) \left[\sum_{i=1}^k \binom{cn}{i} \frac{1}{n^i} - \frac{\varepsilon}{1 - \varepsilon} \binom{cn}{k+1} \frac{1}{n^{k+1}} \right] \\
&\leq \varepsilon^k (1 - \varepsilon) \sum_{i=1}^k \binom{cn}{i} \frac{1}{n^i} \quad \blacktriangleleft
\end{aligned}$$

We simplify the probabilities in Lemma 2 in Corollary 3. The probability bounds in Corollary 3 closely match the distribution of hash-selector frequencies we observe experimentally.

► **Corollary 3.** *Consider a sequence $Q = x_1, x_2, \dots, x_t$ of queries (interleaved with inserts), where each $x_i \notin \mathcal{S}$ and Q consists of cn unique queries (with any number of repetitions), where $c < 1/\varepsilon - 1$. For any $y \in \mathcal{S}$, if $v(y)$ is the hash-selector value of y after all queries in Q are performed, then:*

$$\Pr[v(y) = 0] < \frac{1}{e^{c\varepsilon}}, \text{ and } \Pr[v(y) = k] < \varepsilon^k \sum_{i=1}^k \frac{c^i}{i!} \text{ for } k \geq 1.$$

Proof. To upper bound $\Pr[v(y) = 0]$, we use the inequality $(1 - 1/x)^x \leq 1/e$ for $x > 1$. To upper bound $\Pr[v(y) = k]$, we upper bound:

$$\binom{cn}{i} \frac{1}{n^i} \leq \frac{cn \cdot (cn-1) \cdots (cn-i)}{i!} \frac{1}{n^i} \leq \frac{c^i n^i}{i! n^i} = \frac{c^i}{i!} \quad \blacktriangleleft$$

Space analysis. Up until now, we have assumed that we always have enough room to store arbitrarily large hash selector values. Next, we give a tradeoff between the space usage of the data structure and the number of unique queries it can support.

We use the hash-selector probabilities derived above to analyze the space overhead of storing hash-selector values. Theorem 4 assumes an optimal arithmetic encoding: storing a hash-selector value k that occurs with probability p_k requires exactly $\log(1/p_k)$ bits. In our implementation we use an approximate version of the arithmetic coding for the sake of performance.

► **Theorem 4.** *For any $\varepsilon < 1/2$ and $b \geq 2$, given a sequence of $n/(b\sqrt{\varepsilon})$ unique queries (with no restriction on the number of repetitions of each), the telescoping adaptive filter maintains a sustained false-positive rate of ε using at most $\left(\frac{1}{e} + \frac{b}{(b-1)^2}\right)$ bits of space in expectation per element.*

Proof. Let $c = 1/(b\sqrt{\varepsilon})$; thus, there are cn unique queries. Consider an arbitrary element $y \in \mathcal{S}$. The expected space used to store the hash-selector value $v(y)$ of y is $\sum_{k=0}^{\infty} p_k \log 1/p_k$, where p_k is the probability that $v(y) = k$.

We separate out the case where $k = 0$, for which p_k is the largest, and upper bound the $p_0 \log 1/p_0$ term below, using the probability derived in Lemma 2.

$$\begin{aligned}
p_0 \log 1/p_0 &= (1 - \varepsilon/n)^{cn} \log \frac{1}{(1 - \varepsilon/n)^{cn}} \leq \frac{1}{e^{c\varepsilon}} \cdot \log(1 + \frac{\varepsilon}{n})^{cn} \\
&= \frac{1}{e^{c\varepsilon}} \cdot cn \log(1 + \frac{\varepsilon}{n}) \leq \frac{1}{e^{c\varepsilon}} \cdot cn \frac{\varepsilon}{n} = \frac{c\varepsilon}{e^{c\varepsilon}} < \frac{1}{e} \quad (2)
\end{aligned}$$

In step (2) above we use the fact that $x/e^x < 1/e$ for all $x > 0$.

We now upper bound the rest of the summation, that is, $\sum_{k=1}^{\infty} p_k \log 1/p_k$ for $k \geq 1$. When upper bounding this summation we will be using upper bounds on p_k – but this is a *lower* bound on $\log 1/p_k$. To deal with this, we observe that the function $x \log 1/x$ is monotonically increasing for $x < 1/e$. Therefore, if we show that the bounds in Corollary 3 never exceed $1/e$, we can substitute both terms in $p_k \log 1/p_k$ in our analysis. We start by showing this upper bound. In the following, we use $b \geq 2$ and $\varepsilon < 1/2$.

$$p_k < \varepsilon^k \sum_{i=1}^k \frac{c^i}{i!} < \varepsilon^k c^k \cdot k < \varepsilon^k \cdot \left(\frac{1}{b\sqrt{\varepsilon}} \right)^k \cdot k = \frac{k}{b^k} \cdot \varepsilon^{k/2} < k \cdot \frac{1}{2^{3k/2}} < \frac{1}{e}.$$

We now upper bound the sum $\sum_{k=1}^{\infty} p_k \log 1/p_k$ by replacing p_k with its upper bound $\varepsilon^k c^k \cdot k$ (this replacement is an upper bound because we showed $\varepsilon^k \sum_{i=1}^k \frac{c^i}{i!} < 1/e$ above).

$$\sum_{k=1}^{\infty} p_k \log 1/p_k \leq \sum_{k=1}^{\infty} k \varepsilon^k c^k \log \frac{1}{\varepsilon^k c^k k} = \sum_{k=1}^{\infty} \frac{k}{b^k} \cdot \left(\varepsilon^{k/2} \cdot \log 1/\varepsilon^k \right) \quad (3)$$

$$< \sum_{k=1}^{\infty} \frac{k}{b^k} = \frac{b}{(b-1)^2}. \quad (4)$$

We simplify step (3) above using the fact that $\sqrt{x} \log 1/x < 1$ for all $x \leq 1$; step (4) is a known identity.

Thus, $\sum_{k=0}^{\infty} p_k \log 1/p_k < 1/e + b/(b-1)^2$, which is the expected number of bits used to store the hash-selector value of y . ◀

Theorem 4 implies that if the TAF is using a certain number of bits per element in expectation to store hash-selector values, then there is a precise bound on the number of unique queries it can handle in any query workload while being provably adaptive. For example, if $\varepsilon = 1/2^8$ and we set $b = 4$ in Theorem 4, then a telescoping adaptive filter that uses $4/9 + 1/e \approx 0.812$ bits per element in expectation can handle $4n$ unique queries without running out of space and having to rebuild. In Section 6, the TAF outperforms this bound, retaining good performance with 0.812 bits per element for $A/S \leq 20$.

5 Implementation

In this section, we describe the implementation of the TAF and our implementation of the broom filter [5], which we call the *extension adaptive filter* (exAF).

Recall that adaptive filters have a local state **L** and a remote representation **R**.

Rank-and-select quotient filter. The local state **L** of both the TAF and exAF is implemented as a rank-and-select quotient filter (RSQF) [29]. The RSQF stores metadata bits – one **occupied** bit and one **runend** bit for each slot. The occupied bit associated with slot i indicates whether any elements with the quotient i have been inserted into the filter. The runend bit associated with slot i tracks whether the remainder placed in slot i is the last remainder in a contiguous run of remainders with the same quotient. These metadata bits are sufficient to find the original slot of an element, but processing them bit-by-bit can be slow. The RSQF cleverly uses *rank* and *select* operations to quickly jump to the original slot [6]. These operations are efficiently implemented using x86 instructions on 64-bit words.

To improve cache efficiency, the RSQF stores remainders (along with their 2 metadata bits) in 64-element *blocks*. In particular, each block stores 64 contiguous remainders and two 64-bit metadata arrays. To search through the blocks efficiently, an **offset** (stored using at

most 8 bits) is stored for each block. The offset of a location i is the distance between i and i 's associated runend. Each block stores the offset of its first slot. In total, the RSQF stores 2.125 metadata bits per element in the filter.

Arithmetic coding on integers. Arithmetic coding can give theoretically optimal compression, but the standard implementation that recursively divides the unit interval relies on floating point operations. These floating point operations are slow in practice, and involve precision issues that can lead to incorrect answers or inefficient representations. In our implementation, we avoid these issues by applying arithmetic coding to a range of integers, $\{0, \dots, 2^k - 1\}$ for the desired code length k , instead of the unit interval. We set $k = 56$, encoding all hash-selector values for a block in a 56-bit word. When multiplying or dividing integral intervals by probabilities in $[0, 1]$, we approximate floating point operations using integer shifts and multiplications.

Remote representation. We implement \mathbf{R} for both filters as an array storing elements in the set \mathcal{S} , along with their associated hashes. We keep \mathbf{R} in sync with \mathbf{L} : if the remainder $r(x)$ is stored in slot s in \mathbf{L} , then x is stored in slot s in \mathbf{R} . This leads to easy lookups: to lookup an element x in \mathbf{R} , we simply check the slot $\mathbf{R}[s]$ where $r(x) = \mathbf{L}[s]$. Insertions that cause remainders to shift in \mathbf{L} are expensive, however, as we need to shift elements in \mathbf{R} as well.

TAF implementation. The local state of TAF is an RSQF where each block of 64 contiguous elements stores the remainders of all elements, all metadata bits (each type stored in a 64-bit word), an 8-bit offset, and a 56-bit arithmetic code storing hash-selector values.

TAF's inserts are similar to the RSQF, which may require shifting remainders. The TAF updates the hash-selector values of all blocks that are touched by the insertion.

Our implementation uses MurmurHash [3] which has a 128-bit output. We partition the output of MurmurHash into the quotient, followed by chunks of size $\log(1/\varepsilon)$, where each chunk corresponds to one remainder. Each time we increment the hash-selector value, we just slide over $\log(1/\varepsilon)$ bits to obtain the new remainder.

On a query x , the TAF goes through each slot s corresponding to quotient $q(x)$ and compares the remainder stored in s to $r_i(y)$, where i is the hash-selector value of s , retrieved by decoding the blocks associated with each s . If they match, the filter returns "present" and checks \mathbf{R} to determine if $x \in \mathcal{S}$. If $x \notin \mathcal{S}$, the filter increments the hash-selector i of x and updates the arithmetic code of the block containing x .

If the 56-bit encoding fails, we **rebuild**: we set all hash-selector bits in the block to 0, and then attempt to fix the false positive again.

exAF implementation. Our implementation of the broom filter, which we call the exAF, maintains its local state as a blocked RSQF, similar to the TAF. The main difference between the two filters is how they adapt. The exAF implements the broom filter's adapt policy of lengthening fingerprints. To do this efficiently, we follow a strategy similar to the TAF. We divide the data structure into blocks of 64 elements, storing all extensions for a single block into an arithmetic code that uses at most 56 bits.

The exAF's insertion algorithm resembles the that of the RSQF and broom filter. However, while the broom filter adapts on inserts to ensure that all stored fingerprints are unique, the exAF does not adapt on inserts, and may have duplicate fingerprints.

During a query operation, the exAF first performs an RQSF query: it finds if there is a stored element whose quotient and remainder bits match, without accessing any extension bit. Only if these match does it decode the block's arithmetic code, allowing it to check extension bits. This makes queries in the exAF faster compared to TAF, which must perform decodes on all queries. If the full fingerprint of a query y collides with an element $x \in \mathcal{S}$, the filter returns “present” and checks \mathbf{R} to determine if $x \in \mathcal{S}$. If $x \notin \mathcal{S}$, the exAF adapts by adding extension bits to $f(x)$ by decoding the block's arithmetic code, updating x 's extension bits, and re-encoding.

As in the TAF, if the 56-bit encoding fails, the exAF rebuilds by setting all adaptivity bits in the block to 0, and then attempts to fix the false positive again.

6 Evaluation

In this section, we empirically evaluate the telescoping adaptive filter and the exAF.

We compare the false-positive performance of these filters to the Cuckooing ACF, the Cyclic ACF (with $s = 1, 2, 3$ hash-selector bits), and the Swapping ACF. The Cyclic ACF and the Cuckooing ACF use 4 random hashes to choose the location of each element, and have bins of size 1. The Swapping ACF uses 2 location hashes and bins of size 4.

We compare the throughput of the TAF and exAF against the vacuum filter [32], our implementation of the RSQF, and a space-inefficient version of the TAF that does not perform arithmetic coding operations.

Experimental setup. We evaluate the filters in terms of the following parameter settings.

- *Load factor.* For the false-positive tests, we use a load factor of .95. We evaluate the throughput on a range of load factors.
- *Fingerprint size.* We set the fingerprint size of each filter so that they all use the same amount of space. We use 8-bit remainders for the TAF. Because the TAF has three extra bits per element for metadata and adaptivity, this corresponds to fingerprints of size 11 for the Swapping and Cuckooing ACF, and size $11 - s$ for a Cyclic ACF with s hash-selector bits.
- *A/S ratio.* The parameter A/S (shorthand for $|A|/|\mathcal{S}|$) is the ratio of the number of unique queries in the query set A and the size of the filter's membership set \mathcal{S} . Depending on the structure of the queries, a higher A/S value may indicate a more difficult workload, as “fixed” false positives are separated by a large number of interspersed queries.

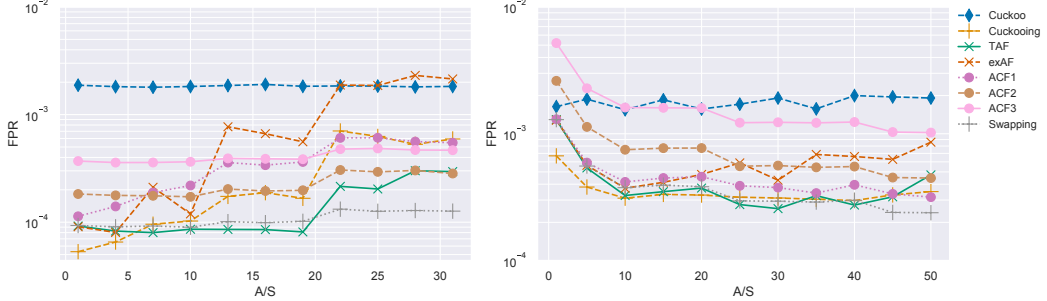
All experiments were run on a workstation with Dual Intel Xeon Gold 6240 18-core 2.6 Ghz processors with 128G memory (DDR4 2666MHz ECC). All experiments were single-threaded.

6.1 False Positive Rate

Firehose benchmark. We measure the false positive rate on data generated by the Firehose benchmark suite [1, 2] which simulates a real-world cybersecurity workload. Firehose has two generators: **power law** and **active set**; we use data from both.

The active set generator generates 64-bit unsigned integers from a continuously evolving “active set” of keys. The probability with which an individual key is sampled varies in time according to a bell-shaped curve to create a “trending effect” as observed in cyberstreams [2]. We generated 10 million queries using the active set generator. We set the value POW_EXP in the active set generator to 0.5 to encourage query repetitions. (Each query is repeated approximately 57 times on average in our final dataset.)

We then generated 50 million queries using the power-law generator, which generates queries using a power-law distribution. This dataset had each query repeated many times; each query was repeated 584 times on average.



■ **Figure 2** False positive rates on the firehose benchmarks. The plot on the left uses the active set generator; the plot on the right uses the power-law generator.

In our tests we vary the size of the stored set S (each uses the same input, so $|A|$ is constant). The results are shown in Figure 2; all data points are the average of 10 experiments. ACF1, ACF2, and ACF3 represent the Cyclic ACF with $s = 1, 2, 3$ respectively.

For the active set generated data, the TAF is the best data structure for moderate A/S . Above $A/S \approx 20$, rebuilds become frequent enough that TAF performance degrades somewhat, after which its performance is similar to that of the Cyclic ACF with $s = 2$ (second to the Swapping ACF). This closely matches the analysis in Section 4.

For the power law data, the TAF is competitive for most A/S values, although again it is best for moderate values.

Notably, in both cases (and particularly for the active set data), the exAF performs substantially worse than the TAF. This shows that given the space amount of extra bits per element on average, the TAF uses them more effectively towards adaptivity than the exAF.

Network Traces. We give experiments on three network trace datasets from the CAIDA 2014 dataset, replicating the experiments of Mitzenmacher et al. [25]. We use three network traces from the CAIDA 2014 dataset, specifically `equinix-chicago.dirA.20140619` (“Chicago A”, Figure 3) `equinixchicago.dirB.20140619-432600` (“Chicago B”, Figure 3), and `equinix-sanjose.dirA.20140320-130400` (“San Jose”, Figure 4).

On network trace datasets, most filters are equally effective at fixing false positives, and their performance is determined mostly by their **baseline false positive rate**, that is, the probability with which a first-time query is a false positive. If s bits are used for adaptivity, that increases the baseline FP rate by 2^s , compared to when those bits are used towards remainders. This gives the Cuckooing ACF an advantage as it uses 0 bits for adapting.

The TAF and exAF perform similarly to the Swapping ACF and ACF1 (Cyclic ACF with $s = 1$) on these datasets.

Adversarial tests. The main advantage of the TAF and exAF is that both are adaptive in theory – even against an adversary. Adversarial inputs are motivated by security concerns, such as denial-of-service attacks, but they may also arise in some situations in practice. For example, it may be that the input stream is performance-dependent, and previous false positives are more likely to be queried again.

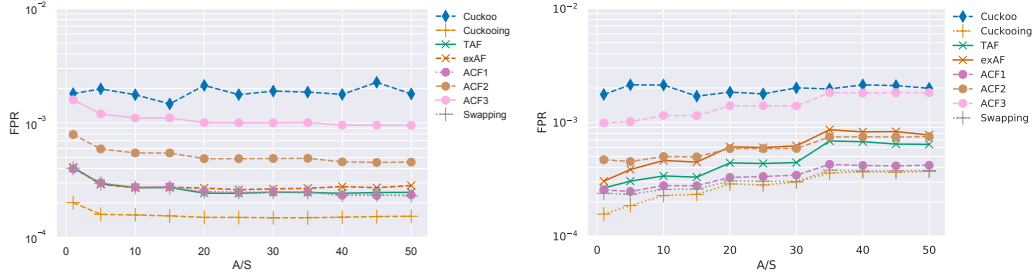


Figure 3 False positive performance of the filters on network trace data. The Chicago A dataset is used on the left, and the Chicago B dataset is on the right.

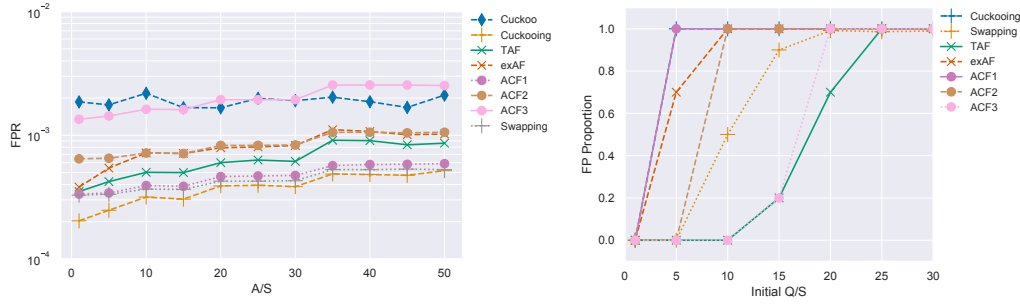


Figure 4 On the left is the network trace San Jose dataset. On the right is adversarial data, where we vary the size of the initial query set, and plot the proportion of elements in the final set that are false positives.

We test our filter against an “adversarial” stream that probabilistically queries previous false positives. This input is significantly simpler than the lower bounds given in [21] and [5], but shares some of the basic structure.

Our adversarial stream starts with a set of random queries $|Q|$. The queries are performed in a sequence of rounds; each divided into 10 subrounds. In a subround, each element of Q is queried. After a round, any element that was never a false positive in that round is removed from Q . The filter then continues to the next round. The test stops when $|Q|/|\mathcal{S}| = .01$, or a bounded number of rounds is reached.

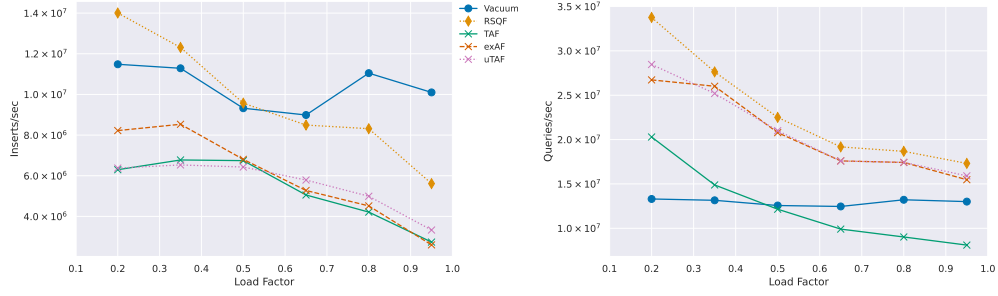
The x-axis of our plot is $|Q|/|\mathcal{S}|$, and the y-axis is the false positive rate during the final round (after the adversary has whittled Q to only contain likely false positives). We again see that the TAF does very well up until around $|Q|/|\mathcal{S}| \approx 20$. After this point, the adversary is successfully able to force false positives. This agrees closely with the analysis in Section 4.

The Cyclic ACF with $s = 3$ (ACF3) does well on adversarial data even though it is known to not be adaptive. This may be in part because the constants in the lower bound proof [21] are very large (the lower bound uses $1/\epsilon^8 \approx 2^{64}$ queries). However, this adaptivity comes at the cost of a worse baseline FP rate, as this filter struggles on network trace data.

6.2 Throughput

In this section, we compare the throughput of our filters to other similar filters.

For the throughput tests, we introduce several new filters as a point of comparison. The vacuum filter [32] is a cuckoo filter variant designed to be space- and cache-efficient. We compare to the “from scratch” version of their filter [34]. We also compare to our implementation of the RSQF [29]. The RSQF does not adapt, or perform remote accesses.



■ **Figure 5** The throughput for inserts (left) and queries (right) on the active set Firehose data.

Finally, to isolate the cost of the arithmetic coding itself, we compare to our implementation of an **uncompressed telescoping adaptive filter (uTAF)**. The uTAF works exactly as the TAF, except it stores its hash-selector values explicitly, without using an arithmetic coding. This means that the uTAF is very space-inefficient.

For the throughput tests, we evaluated the performance on the active set Firehose data used in Figure 2. Our filters used 2^{24} slots. We varied the load factor to compare performance. All data points shown are the average of 10 runs.

The throughput tests show that the TAF achieves similar performance in inserts to the other filters, though it lags behind in queries at high throughput. The exAF performs significantly better for queries, likely due to skipping decodes as discussed in Section 5.

The uTAF is noticeably faster than the TAF, but is similar in performance to exAF. This highlights the trade-offs between the two ways to achieve adaptivity: the exAF scheme of lengthening remainders has better throughput but worse adaptivity per bit; while the TAF scheme of updating remainders has better adaptivity per bit but worse throughput. Overall, while the query-time decodes of TAF do come at a throughput cost, they stop short of dominating performance.

7 Conclusion

We provide a new provably-adaptive filter, the telescoping adaptive filter, that was engineered with space- and cache-efficiency and throughput in mind. The TAF is unique among adaptive filters in that it only uses a fractional number of extra bits for adaptivity (0.875 bits per element). To benchmark the TAF, we also provide a practical implementation of the broom filter. To effectively compress the adaptivity metadata for both filters, we implement arithmetic coding that is optimized for the probability distributions arising in each filter.

We empirically evaluate the TAF and exAF against other state-of-the-art filters that adapt, on a variety of datasets. Our experiments show that TAF outperforms the exAF significantly on false-positive performance, and frequently matches or outperforms other heuristically adaptive filters. Our throughput tests show that our adaptive filters achieve a comparable throughput to their non-adaptive counterparts.

We believe that our technique to achieve adaptivity through variable-length fingerprints is universal and can be used alongside other filters that stores fingerprints of elements (e.g., a cuckoo or vacuum filter). Thus, there is potential for further improvements by applying our ideas to other filters, taking advantage of many years of filter research.

References

- 1 Karl Anderson and Steve Plimpton. Firehose streaming benchmarks. Technical report, Sandia National Laboratory, 2015.
- 2 Karl Anderson and Steve Plimpton. FireHose streaming benchmarks. www.firehose.sandia.gov. Accessed: 2018-12-11.
- 3 Austin Appleby. Murmurhash. <https://github.com/aappleby/smhasher>, 2016. Accessed: 2020-08-01.
- 4 Michael A Bender, Rathish Das, Martín Farach-Colton, Tianchi Mo, David Tench, and Yung Ping Wang. Mitigating false positives in filters: to adapt or to cache? In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 16–24. SIAM, 2021.
- 5 Michael A Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *Symposium on Foundations of Computer Science (FOCS)*, pages 182–193. IEEE, 2018.
- 6 Michael A Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C Kuszmaul, Dzejlja Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don’t thrash: how to cache your hash on flash. *Proc. VLDB Endowment*, 5(11):1627–1637, 2012.
- 7 Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- 8 Alex D Breslow and Nuwan S Jayasena. Morton filters: faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. *Proc. VLDB Endowment*, 11(9):1041–1055, 2018.
- 9 Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.
- 10 J Bruck, Jie Gao, and Anxiao Jiang. Weighted bloom filter. In *Symposium on Information Theory*. IEEE, 2006.
- 11 Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Symposium on Theory of Computing (STOC)*, pages 59–65. ACM, 1978.
- 12 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *Transactions on Computer Systems*, 26(2):4, 2008.
- 13 Saar Cohen and Yossi Matias. Spectral bloom filters. In *International Conference on Management of Data (SIGMOD)*, pages 241–252. ACM, 2003.
- 14 Kyle Deeds, Brian Hentschel, and Stratos Idreos. Stacked filters: learning to filter by structure. *Proc. VLDB Endowment*, 14(4):600–612, 2020.
- 15 Fan Deng and Davood Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *International Conference on Management of Data (SIGMOD)*, pages 25–36. ACM, 2006.
- 16 Peter C Dillinger and Stefan Walzer. Ribbon filter: practically smaller than bloom and xor. *arXiv preprint arXiv:2103.02515*, 2021.
- 17 David Eppstein, Michael T Goodrich, Michael Mitzenmacher, and Manuel R Torres. 2-3 cuckoo filters for faster triangle listing and set intersection. In *Principles of Database Systems (PODS)*, pages 247–260. ACM, 2017.
- 18 Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Conference on emerging Networking Experiments and Technologies (CoNEXT)*, pages 75–88. ACM, 2014.
- 19 Thomas Mueller Graf and Daniel Lemire. Xor filters: Faster and smaller than bloom and cuckoo filters. *Journal of Experimental Algorithmics (JEA)*, 25:1–16, 2020.
- 20 Paul G. Howard and Jeffrey Scott Vitter. *Practical Implementations of Arithmetic Coding*, pages 85–112. Springer US, Boston, MA, 1992. doi:10.1007/978-1-4615-3596-6_4.
- 21 Tsvi Kopelowitz, Samuel McCauley, and Eli Porat. Support optimality and adaptive cuckoo filters. In *Proc. 17th Algorithms and Data Structures Symposium (WADS)*, 2021. To appear.

- 22 Harald Lang, Thomas Neumann, Alfons Kemper, and Peter Boncz. Performance-optimal filtering: Bloom overtakes cuckoo at high throughput. *Proc. VLDB Endowment*, 12(5):502–515, 2019.
- 23 Yoshinori Matsunobu, Siying Dong, and Herman Lee. Myrocks: LSM-tree database storage engine serving Facebook’s social graph. *Proc. VLDB Endowment*, 13(12):3217–3230, 2020.
- 24 Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 462–471, 2018.
- 25 Michael Mitzenmacher, Salvatore Pontarelli, and Pedro Reviriego. Adaptive cuckoo filters. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 36–47. SIAM, 2018.
- 26 Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.
- 27 Patrick O’Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O’Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.
- 28 Anna Pagh, Rasmus Pagh, and S Srinivasa Rao. An optimal bloom filter replacement. In *Symposium on Discrete Algorithms (SODA)*, pages 823–829. ACM-SIAM, 2005.
- 29 Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *International Conference on Management of Data (SIGMOD)*, pages 775–787. ACM, 2017.
- 30 Jack Rae, Sergey Bartunov, and Timothy Lillicrap. Meta-learning neural bloom filters. In *International Conference on Machine Learning (ICML)*, pages 5271–5280. PMLR, 2019.
- 31 Sasu Tarkoma, Christian Esteve Rothenberg, Eemil Lagerspetz, et al. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1):131–155, 2012.
- 32 Minmei Wang and Mingxun Zhou. Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters. *Proc. VLDB Endowment*, 2019.
- 33 Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.
- 34 Mingxun Zhou. Vacuum filter. <https://github.com/wuwuz/Vacuum-Filter>, 2020. Accessed: 2020-12-01.

Finding an Approximate Mode of a Kernel Density Estimate

Jasper C.H. Lee ✉

Brown University, Providence, RI, USA

Jerry Li ✉

Microsoft Research, Redmond, WA, USA

Christopher Musco ✉

New York University, NY, USA

Jeff M. Phillips ✉

University of Utah, Salt Lake City, UT, USA

Wai Ming Tai ✉

University of Chicago, IL, USA

Abstract

Given points $P = \{p_1, \dots, p_n\}$ subset of \mathbb{R}^d , how do we find a point x which approximately maximizes the function $\frac{1}{n} \sum_{p_i \in P} e^{-\|p_i - x\|^2}$? In other words, how do we find an approximate mode of a Gaussian kernel density estimate (KDE) of P ? Given the power of KDEs in representing probability distributions and other continuous functions, the basic mode finding problem is widely applicable. However, it is poorly understood algorithmically. We provide fast and provably accurate approximation algorithms for mode finding in both the low and high dimensional settings. For low (constant) dimension, our main contribution is a reduction to solving systems of polynomial inequalities. For high dimension, we prove the first dimensionality reduction result for KDE mode finding. The latter result leverages Johnson-Lindenstrauss projection, Kirschbraun's classic extension theorem, and perhaps surprisingly, the mean-shift heuristic for mode finding. For constant approximation factor these algorithms run in $O(n(\log n)^{O(d)})$ and $O(nd + (\log n)^{O(\log^3 n)})$, respectively; these are proven more precisely as a $(1 + \epsilon)$ -approximation guarantee. Furthermore, for the special case of $d = 2$, we give a combinatorial algorithm running in $O(n \log^2 n)$ time. We empirically demonstrate that the random projection approach and the 2-dimensional algorithm improves over the state-of-the-art mode-finding heuristics.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Kernel density estimation, Dimensionality reduction, Coresets, Means-shift

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.61

Funding Jasper C.H. Lee: Partially supported by NSF award IIS-1562657.

Christopher Musco: Partially supported by NSF CCF-2045590.

Jeff M. Phillips: Partially supported by NSF CCF-1350888, CNS-1514520, CNS-1564287, IIS-1619287, IIS-1816149, and CBET-1953350.

1 Introduction

Given a point set P in \mathbb{R}^d and a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, the kernel density estimate (KDE) is a function mapping from \mathbb{R}^d to \mathbb{R} and is defined as $\frac{1}{|P|} \sum_{p \in P} K(x, p)$ for any $x \in \mathbb{R}^d$. One common example of kernel K is the Gaussian kernel, $K(x, y) = e^{-\|x - y\|^2}$ for any $x, y \in \mathbb{R}^d$, which is the focus of this paper.

These kernel density estimates are a fundamental tool in statistics [48, 45, 18, 19] and machine learning [44, 23, 36]. For $d = 1$, KDEs with a triangular kernel ($K(x, p) = \max(0, 1 - |x - p|)$) can be seen as the average over all shifts of a fix-width histogram. And



© Jasper C.H. Lee, Jerry Li, Christopher Musco, Jeff M. Phillips, and Wai Ming Tai; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 61; pp. 61:1–61:19

Leibniz International Proceedings in Informatics



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

unlike histograms these generalize naturally to a higher dimensions as a stable way to create a continuous function to represent the measure of a finite point set. Indeed, the KDEs constructed on an iid sample from any tame distribution will converge to that distribution in the limit as the sample size grows [48, 45]. Not surprisingly they are also central objects in Bayesian data analysis [27, 21]. Using Gaussian kernels (and other positive definite kernels), KDEs are members of a reproducing kernel Hilbert space [53, 50, 36] where for instance they induce a natural distance between distributions [49, 28]. Their other applications includes outlier detection [56], clustering [43], topological data analysis [40, 14], spatial anomaly detection [2, 24], and statistical hypothesis testing [23].

In this paper, we study how to find an *approximate mode* of a Gaussian KDE. An ϵ -approximate mode of a KDE is a point x' whose KDE value is at least $1 - \epsilon$ times the maximum of the KDE. It is known that Gaussian KDEs can have complex structure of local maximum [20, 26], but other than some heuristic approaches [9, 10, 54, 22] there has been very little prior work [40, 2] (which we discuss shortly) in developing and formally analyzing algorithms to find this maximum. Beyond being a key descriptor (the mode) of one of the most common representations of a continuous distribution, finding the (global) maximum of a KDE has many other specific applications. It is a necessary step to create a simplicial complex to approximate superlevel sets of KDEs [40]; to localize and track objects [13, 46]; to quantify multi-modality of distributions [47]; to finding typical objects, including curves [22].

Problem Definition. For any $x, y \in \mathbb{R}^d$, we define the Gaussian kernel as $K(x, y) = e^{-\|x-y\|^2}$. The Gaussian kernel density estimate (KDE) $\bar{\mathcal{G}}_P(x)$ of a point set P is defined as $\bar{\mathcal{G}}_P(x) = \frac{1}{|P|} \sum_{p \in P} K(p, x)$, for $x \in \mathbb{R}^d$. We will sometimes use the notation $\mathcal{G}_P(x) = |P| \cdot \bar{\mathcal{G}}_P(x)$ to simplify calculations. In line with other works on optimization, we focus on the approximate version of the mode finding problem, defined as follows. Given a point set P of size n where $\max_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x) \geq \rho$ for some parameter ρ below which the maximum is uninteresting, and an error parameter $\epsilon > 0$, the goal is to find an ϵ -approximate mode x' , such that $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon) \max_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x)$. We assume the lower bound ρ is known to the algorithm; or we can set $\rho = 1/n$ since $\bar{\mathcal{G}}_P(p) \geq 1/n$ for any $p \in P$. In practice, one should expect that $\rho \ll \epsilon$, so we aim for algorithms with far smaller dependence on $1/\rho$ than on $1/\epsilon$.

Known Results. One trivial approach is exhaustive search. It is easy to see that the optimal point x^* cannot be too far away from the input data. More precisely, x^* should be within the radius of $\sqrt{\log \frac{1}{\rho}}$ of a point p for some $p \in P$. Given the above observation, one can construct a grid of width $\frac{1}{\epsilon\rho}$ around each point of input data and evaluate the value of \mathcal{G}_P at each grid point. This approach will allow us to output a solution with additive error at most $\epsilon n \rho$. However, the size of the search space could be as large as $O\left(n\left(\sqrt{\log \frac{1}{\rho}}/\epsilon\rho\right)^d\right)$ which is infeasible in practice. A similar approach is suggested by [40].

Another approach, proposed by [2], is to compute the depth in an arrangement of a set of geometric objects. Namely, it is to find the point that maximizes the number of objects including that point. They consider a set \mathcal{S} of segment in \mathbb{R}^2 and, for any $x \in \mathbb{R}^2$ and $s \in \mathcal{S}$, define $K(x, s) = K(x, y)$ where y is the closest point on s to x . In our setting, we treat the point set P as degenerate (length 0) segments \mathcal{S} . By discretizing the continuous function K into the level set of it, one can view the problem as computing the depth in an arrangement of a collection of level sets. This approach has a running time of $O(\frac{n}{\epsilon^2} \log^3 n)$ (this implicitly sets $\rho = 1/n$). One can generalize their approach to the high dimensional case, but the running time would still be $O(n^{O(d)})$.

Related Work. As mentioned before, computing depth in an arrangement of a set of geometric object is highly related to our problem. Given a collection \mathcal{C} of geometric object in \mathbb{R}^d , one can express the depth as $\sum_{c \in \mathcal{C}} \mathbf{1}_c(x)$ where $\mathbf{1}_c(x)$ is the indicator function of $x \in c$. It is easy to see that KDE is basically the same formula by replacing $\mathbf{1}_c(x)$ with $K(x, p)$. Namely, one can view finding a maximum point of KDE as computing the point of maximum “fractional” depth among kernels. Surprisingly, there are not many non-trivial algorithmic results on computing the depth of high-dimensional geometric objects. In general, whenever \mathcal{C} is a collection of bounded complexity (e.g. VC dimension [52]) objects, the arrangement is always of complexity $O(n^{O(d)})$ and it can be constructed, with the depth encoded, in as much time. A celebrated case is when \mathcal{C} is a collection of axis parallel box, the point of maximum depth can be found in $O(n^{d/2 - o(1)})$ time [12]. For our task we can run such approaches on a sample of size $n_0 = O(\frac{d}{\epsilon^2 \rho} \log \frac{1}{\rho})$ [32, 25], so the runtimes still have a $1/\rho^{O(d)}$ term.

Another line of work [11, 20, 5] attempts to bound the *number* of local maximum of a Gaussian KDE. Perhaps surprisingly, the number is greater than n for dimensions $d \geq 2$, and in fact can be at least $\binom{n}{d} + n$ for $n, d \geq 2$ [5]. It is currently unknown whether the number can be infinite, but the best upper bound *assuming finiteness* is $2^{d + \binom{n}{2}}(5 + 3d)^n$. So even if we could identify all of these, there still would be $\Omega(n^d)$ points to evaluate.

While not as explicit as the dimensionality reduction results we will present, other work based on LSH [6, 7] or related to geometric graphs [41] have shown properties of evaluating KDEs after what can be interpreted as forms of dimensionality reduction. For instance, Quanrud [41] shows that using dimensionality reduction, as well as other approximations and structures, one can evaluate KDEs within $1 \pm \epsilon$ error in roughly $1/\epsilon^2$ time, but also with logarithmic factors depending on, for example, spread parameters for the Gaussian kernel.

Our Approach and Result. We present an approximation scheme that reads the data (to sample it) in $O(nd)$ time, and then its runtime depends only on $1/\epsilon$ and $1/\rho$. At the heart of our algorithm are two techniques: dimensionality reduction and polynomial system solving. We also use standard coresets results for Gaussian kernel density estimates.

For dimensionality reduction (Section 3), we use Johnson-Lindenstrauss matrices to project the point set down to low dimensions, and solve the problem in low dimensions. The crucial issue is, if we solve the mode finding problem in the low dimensional space, it is not immediately clear that the original high dimensional space also has a point that gives a high KDE value. We resolve this with an application of Kirschbraun’s extension theorem [30, 51], which shows the existence of such a high dimensional point. To find the actual point in the high dimensional space, we use one step of the *mean-shift* algorithm [9, 10], which is a known heuristic for the KDE maximum finding problem with provable monotonicity properties. We could alternatively combine a *terminal dimensionality reduction* result [37] with our mean-shift recovery strategy. Doing so would give the same level of dimensionality reduction, at the expense of reduced simplicity and runtime efficiency.

In low dimensions, we consider Taylor series truncations of the Gaussian kernel, and reduce the mode finding problem to solving systems of polynomial inequalities (Section 2). The result of [42] implies that one can find a solution to a system of λ polynomial inequalities with degree D and k variables in time $O((\lambda D)^{O(k)})$. Here, k will essentially be the dimensionality d of the problem, and λ will be a constant as shown in our constructions. We observe that since the optimal point must be close to one of the points in the input, we can consider a sufficiently fine grid in the vicinity of each input point, which totals to $O(n2^{O(d)})$ grid points. For each grid point, we formulate and solve a system of polynomial inequalities based on Taylor expansions, up to $O(\log \frac{1}{\rho})$ terms around that grid point. This gives a running time of $O(n(\log \frac{1}{\rho})^{O(d)})$, where n is the size of the input point set.

Combining the above ideas with standard coresets results (small subsets $Q \subset P$ so $\bar{\mathcal{G}}_Q$ approximates $\bar{\mathcal{G}}_P$) yields approximation schemes for the KDE mode finding problem. We present two such schemes, one with exponential runtime dependence on the dimensionality d which is more suitable for low dimensions, and another with only linear dependence in d (which is necessary for reading the input) and is designed for the high dimensional regime. Guarantees of these approximations are captured by Theorems 1 and 2; we provide algorithmic details and intuition, but proofs are in the appendices.

► **Theorem 1 (Low dimensional regime).** *Given $\epsilon, \rho > 0$ and a point set $P \subset \mathbb{R}^d$ of size n with $\bar{\mathcal{G}}_P(x^*) \geq \rho$, where $x^* = \operatorname{argmax}_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x)$, we can find $x' \in \mathbb{R}^d$ so $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*)$ in $O\left(nd + \frac{d}{\epsilon^2 \rho} \cdot \log \frac{1}{\rho \delta} \cdot \left(\log \frac{d}{\epsilon \rho}\right)^{O(d)}\right)$ time with probability at least $1 - \delta$.*

► **Theorem 2 (High dimensional regime).** *Given $\epsilon, \rho > 0$ and a point set $P \subset \mathbb{R}^d$ of size n with $\bar{\mathcal{G}}_P(x^*) \geq \rho$, where $x^* = \operatorname{argmax}_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x)$, we can find $x' \in \mathbb{R}^d$ so $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*)$ in $O\left(nd + \left(\log \frac{1}{\epsilon \rho}\right)^{O(\frac{1}{\epsilon^2} \log^3 \frac{1}{\epsilon \rho})} \cdot \log \frac{1}{\delta} + \min\{nd \log \frac{1}{\delta}, \frac{d}{\epsilon^2 \rho^2} \log^2 \frac{1}{\delta}\}\right)$ time with probability at least $1 - \delta$.*

One may set the relative error parameter ϵ , and failure probability δ to constants, and observe the mode of $\bar{\mathcal{G}}_P$ must be at least $1/n$ and set $\rho = 1/n$. Then the runtimes become $O(n(\log n)^{O(d)})$ for constant dimensions, and $O(nd + (\log n)^{O(\log^3 n)})$ in high dimensions.

In addition to our result in Theorems 1 and 2, we also consider the special case where $d = 2$. We present a combinatorial algorithm for the 2-dimensional regime which is easier to implement. Here, we borrow the idea from [2] which is to compute the depth. Instead of simply considering the level sets of the Gaussian kernel (which are circles in our setting), we consider a more involved decomposition. One important property of the Gaussian kernel is its multiplicatively separability – namely, the Gaussian kernel can be decomposed into factors, with one factor for each dimension. We now discretize each factor into level sets (which are simply intervals) and then consider their Cartesian products, generating a collection of axis-parallel rectangles. A similar idea was also suggested by [38]. Finally, if we compute the depth of this collection of axis-parallel rectangles, we can find out an approximate mode in time $O(\frac{1}{\epsilon^2 \rho} \log^2 \frac{1}{\rho})$. This approach also works in higher dimensions, but it would yield a slower running time than our general approaches in Theorems 1 and 2. The formal guarantees of this 2-d algorithm are captured in Theorem 3, and proven in the appendices.

► **Theorem 3 (2-dimensional setting).** *Given $\epsilon, \rho > 0$ and a point set $P \subset \mathbb{R}^2$ of size n such that $\bar{\mathcal{G}}_P(x^*) \geq \rho$, where $x^* = \operatorname{argmax}_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x)$, we can find $x' \in \mathbb{R}^2$ so $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*)$ in $O\left(n + \frac{1}{\epsilon^2 \rho} (\log \frac{1}{\rho} + \log \frac{1}{\delta}) \log(\frac{1}{\epsilon \rho} \log \frac{1}{\delta})\right)$ time with probability at least $1 - \delta$.*

There are different extensions of our problem formulation. For example, one can define the weighted KDE of a point set P , $\sum_{p \in P} w_p K(x, p)$, and find its mode. Another common extension is to consider non-spherical Gaussians with different variances. We expect that our technique with some straightforward modifications work for these extensions and will omit the details.

2 KDE Mode Finding via System of Polynomials

In this section we provide algorithms that approximately find the maximum of the Gaussian KDE in \mathbb{R}^d . We first define the following notations. For a point $p \in \mathbb{R}^d$ and $r > 0$, we define $B_p(r)$ as $\{y \in \mathbb{R}^d \mid \|y - p\| \leq r\}$, namely the Euclidean ball around p . For a

point set $P \subset \mathbb{R}^d$ and $r > 0$, we define $B_P(r)$ as $\cup_{p \in P} B_p(r)$, that is the union of Euclidean balls around all the points in P . For a point set $P \subset \mathbb{R}^d$, a point $q \in \mathbb{R}^d$ and $r > 0$, also define $Q_{P,q}(r) = P \cap B_q(r\sqrt{\log \frac{1}{\epsilon\rho}})$. Finally, let $\text{Grid}(\gamma)$ be the infinite grid $\{x = (i_1\gamma, i_2\gamma, \dots, i_d\gamma) \mid i_1, i_2, \dots, i_d \text{ are integers}\}$, parametrized by a cell length $\gamma > 0$.

We first make an observation that the maximum point must be close to one of the data points, captured by Observation 4.

► **Observation 4.** $x^* \in B_P(\sqrt{\log \frac{1}{\rho}})$. Recall that $x^* = \operatorname{argmax}_{x \in \mathbb{R}^d} \mathcal{G}_P(x)$.

Proof. Suppose $x^* \notin B_P(\sqrt{\log \frac{1}{\rho}})$. Then, $\mathcal{G}_P(x^*) = \sum_{p \in P} e^{-\|p-x^*\|^2} < \sum_{p \in P} \rho = n\rho$. However, $\mathcal{G}_P(x^*) \geq n\rho$ by assumption. ◀

The algorithm presented in this section relies crucially on the result of Renegar for solving systems of polynomial inequalities, as stated in the following lemma.

► **Lemma 5** ([42]). *Consider λ polynomial inequalities with maximum degree D and k variables. There is an algorithm either finds a solution that satisfies all λ polynomial inequalities or returns NO SOLUTION in $O((\lambda D)^{O(k)})$ time.*

Before we give details of our algorithm, we present the family of systems of polynomial inequalities we formulate for mode finding. Let $\text{SysPoly}(P, q, r, r', \beta)$ be the following system.

$$\sum_{p \in Q_{P,q}(r')} \prod_{i=1}^d \left(\sum_{j=0}^{s-1} \frac{1}{j!} (-(x_i - p_i)^2)^j \right) \geq \beta \quad \bigwedge \quad \|x - q\|^2 \leq r^2 \log \frac{1}{\epsilon\rho}$$

where $s = (r + r')^2 e^2 \log \frac{d}{\epsilon\rho}$. Intuitively, if a point $x \in \mathbb{R}^d$ satisfies the left inequality of $\text{SysPoly}(P, q, r, r', \beta)$, then the value $\mathcal{G}_P(x)$ is larger than a threshold that is approximately β . It is because the LHS of the left inequality is the sum of the truncated Taylor expansion of the Gaussians centered at p that is around q . On the other hand, the truncated Taylor expansion only gives a good approximation locally. Hence, the right inequality of $\text{SysPoly}(P, q, r, r', \beta)$ ensures that x is around q .

Also, let $\text{SysPoly}(P, q, r, r')$ be the algorithm that performs binary search on β of the above system $\text{SysPoly}(P, q, r, r', \beta)$ and terminates when the search gap is less than $\frac{1}{10} |P| \epsilon\rho$. Note that β lies between 0 and $O(|P|)$ which means we need $O(\log(|P| / \frac{1}{10} |P| \epsilon\rho)) = O(\log \frac{1}{\epsilon\rho})$ iterations in binary search. The total running time of $\text{SysPoly}(P, q, r, r')$ is $O((4s)^{O(d)} \log \frac{1}{\epsilon\rho}) = O(s^{O(d)})$ since $k = d$, $\lambda = 2$ and $D = 2s$ in Lemma 5.

The following lemma captures the approximation error from the Taylor series truncation.

► **Lemma 6.** *Suppose $r + r' > 1$ and $q \in \mathbb{R}^d$ such that $\|x^* - q\| \leq r\sqrt{\log \frac{1}{\epsilon\rho}}$. Then, the output $x^{(q)}$ of $\text{SysPoly}(P, q, r, r')$ satisfies $\mathcal{G}_{Q_{P,q}(r')}(x^{(q)}) \geq \mathcal{G}_{Q_{P,q}(r')}(x^*) - |P| \frac{\epsilon\rho}{2}$.*

In short, it shows that the truncation of the above infinite summation of polynomial terms (wrapped in a sum over all points Q , and the product over d dimensions) induces an error terms $\mathcal{E}(x^{(q)})$ and $\mathcal{E}(x^*)$ at $x^{(q)}$ and x^* , respectively. We can show that the difference between these terms is at most $\epsilon\rho$ for our choice of s , as desired.

61:6 Finding an Approximate Mode of a Kernel Density Estimate

Proof. First, we write $\sum_{p \in Q_q(r')} e^{-\|p - x^{(q)}\|^2}$ into the following form.

$$\begin{aligned} \sum_{p \in Q_q(r')} e^{-\|p - x^{(q)}\|^2} &= \sum_{p \in Q_q(r')} \prod_{i=1}^d \left(\sum_{j=0}^{\infty} \frac{1}{j!} \left(-(x_i^{(q)} - p_i)^2 \right)^j \right) \\ &= \sum_{p \in Q_q(r')} \prod_{i=1}^d \left(\sum_{j=0}^s \frac{1}{j!} \left(-(x_i^{(q)} - p_i)^2 \right)^j \right) + \mathcal{E}(x^{(q)}) \end{aligned}$$

where $\mathcal{E}(x) = \sum_{p \in Q_q(r')} \sum_{j_1, \dots, j_d | \text{one of } j_i \geq s} \frac{1}{j_1! \dots j_d!} \left(-(x_1 - p_1)^2 \right)^{j_1} \dots \left(-(x_d - p_d)^2 \right)^{j_d}$ for any $x \in \mathbb{R}^d$.

Now, we have

$$\begin{aligned} \sum_{p \in Q_q(r')} e^{-\|x^{(q)} - p\|^2} &= \sum_{p \in Q_q(r')} \prod_{i=1}^d \left(\sum_{j=0}^s \frac{1}{j!} \left(-(x_i^{(q)} - p_i)^2 \right)^j \right) + \mathcal{E}(x^{(q)}) \\ &\geq \sum_{p \in Q_q(r')} \prod_{i=1}^d \left(\sum_{j=0}^s \frac{1}{j!} \left(-(x_i^* - p_i)^2 \right)^j \right) - |P| \frac{\epsilon \rho}{10} + \mathcal{E}(x^{(q)}) \\ &\geq \sum_{p \in Q_q(r')} e^{-\|x^* - p\|^2} - |P| \frac{\epsilon \rho}{10} + \mathcal{E}(x^{(q)}) - \mathcal{E}(x^*) \end{aligned}$$

In order to analyze the term $\mathcal{E}(x^{(q)})$ and $\mathcal{E}(x^*)$, we can first analyze the term

$$\left| \sum_{j_1, \dots, j_d | \text{one of } j_i \geq s} \frac{1}{j_1! \dots j_d!} \alpha_1^{j_1} \dots \alpha_d^{j_d} \right|$$

where $\alpha_i = -(y_i - p_i)^2$ where y is $x^{(q)}$ or x^* .

$$\sum_{j_1, \dots, j_d | \text{one of } j_i \geq s} \left(\prod_{i=1}^d \frac{1}{j_i!} \alpha_i^{j_i} \right) = \sum_{i=1}^d \left(\prod_{k=1}^{i-1} \sum_{j=0}^{s-1} \frac{1}{j!} \alpha_k^j \right) \left(\sum_{j=s}^{\infty} \frac{1}{j!} \alpha_i^j \right) \left(\prod_{k=i+1}^d \sum_{j=0}^{\infty} \frac{1}{j!} \alpha_k^j \right)$$

For each $i = 1, 2, \dots, d$, by taking $s = (r + r')^2 e^2 \log \frac{d}{\epsilon \rho}$,

$$\left| \sum_{j=s}^{\infty} \frac{1}{j!} \alpha_i^j \right| \leq \sum_{j=s}^{\infty} \frac{1}{j!} |\alpha_i|^j \leq \max_{\xi \in [-|\alpha_i|, |\alpha_i|]} \frac{e^{\xi}}{s!} |\alpha_i|^s$$

The last inequality is the error approximation of Taylor expansion of exponential function.

Note that $|\alpha_i| = (y_i - p_i)^2 \leq \|y - p\|^2 \leq (\|y - q\| + \|p - q\|)^2 \leq \left(r \sqrt{\log \frac{1}{\epsilon \rho}} + r' \sqrt{\log \frac{1}{\epsilon \rho}} \right)^2 \leq (r + r')^2 \log \frac{1}{\epsilon \rho}$. We have

$$\begin{aligned} \left| \sum_{j=s}^{\infty} \frac{1}{j!} \alpha_i^j \right| &\leq \frac{e^{(r+r')^2 \log \frac{1}{\epsilon \rho}}}{s!} ((r + r')^2 \log \frac{1}{\epsilon \rho})^s \\ &\leq \frac{e^{(r+r')^2 \log \frac{1}{\epsilon \rho}}}{s^s} ((r + r')^2 e \log \frac{1}{\epsilon \rho})^s && \text{by } s! \geq \left(\frac{s}{e}\right)^s \\ &\leq \frac{e^{(r+r')^2 \log \frac{1}{\epsilon \rho}}}{e^s} \leq \left(\frac{\epsilon \rho}{d}\right)^{(r+r')^2 (e^2 - 1)} && \text{recall that } s = (r + r')^2 e^2 \log \frac{d}{\epsilon \rho} \\ &\leq \frac{\epsilon \rho}{20d} && \text{by } r + r' > 1 \text{ and for sufficient small } \epsilon \rho \end{aligned}$$

Now, we can plug this into $\left| \sum_{j_1, \dots, j_d | \text{one of } j_i \geq s} \frac{1}{j_1! \dots j_d!} \alpha_1^{j_1} \dots \alpha_d^{j_d} \right|$.

$$\begin{aligned}
& \left| \sum_{j_1, \dots, j_d | \text{one of } j_i \geq s} \frac{1}{j_1! \dots j_d!} \alpha_1^{j_1} \dots \alpha_d^{j_d} \right| \\
&= \left| \sum_{i=1}^d \left(\prod_{k=1}^{i-1} \sum_{j=0}^{s-1} \frac{1}{j!} \alpha_k^j \right) \left(\sum_{j=s}^{\infty} \frac{1}{j!} \alpha_i^j \right) \left(\prod_{k=i+1}^d \sum_{j=0}^{\infty} \frac{1}{j!} \alpha_k^j \right) \right| \\
&\leq \sum_{i=1}^d \left(\prod_{k=1}^{i-1} \left(1 + \frac{\epsilon \rho}{10d} \right) \right) \left(\frac{\epsilon \rho}{10d} \right) \left(\prod_{k=i+1}^d e^{\alpha_k} \right) \\
&\leq \left(1 + \frac{\epsilon \rho}{20d} \right)^d \frac{\epsilon \rho}{20} \leq e^{\frac{\epsilon \rho}{20}} \frac{\epsilon \rho}{20} \leq \frac{\epsilon \rho}{8} \quad \text{for sufficient small } \epsilon \rho
\end{aligned}$$

That means

$$\begin{aligned}
\sum_{p \in Q_q(r')} e^{-\|x^{(q)} - p\|^2} &\geq \sum_{p \in Q_q(r')} e^{-\|x^* - p\|^2} - |P| \frac{\epsilon \rho}{10} + \mathcal{E}(x^{(q)}) - \mathcal{E}(x^*) \\
&\geq \sum_{p \in Q_q(r')} e^{-\|x^* - p\|^2} - |P| \frac{\epsilon \rho}{10} - |Q_{P,q}(r')| \frac{\epsilon \rho}{8} - |Q_{P,q}(r')| \frac{\epsilon \rho}{8} \\
&= \sum_{p \in Q_q(r')} e^{-\|x^* - p\|^2} - |P| \frac{\epsilon \rho}{2}. \quad \blacktriangleleft
\end{aligned}$$

2.1 Algorithm for Searching Polynomial Systems in Neighborhoods

A first attempt invokes Lemma 6 in a ball $B_p \left(\sqrt{\log \frac{1}{\rho}} \right)$ around each $p \in P$. However, to find out the subset of points that lie inside the ball $B_p \left(\sqrt{\log \frac{1}{\rho}} \right)$ for each $p \in P$, one needs to search linearly over P naively. Therefore, it requires $\Omega(n^2)$ runtime.

Rather, following Algorithm 1, we create a set \mathbf{G}_P of neighborhoods, defined by the subset of $\text{Grid} \left(2\sqrt{\frac{\log \frac{1}{\epsilon \rho}}{d}} \right)$ which is within $4\sqrt{\log \frac{1}{\epsilon \rho}}$ of some point $p \in P$. For each $q \in \mathbf{G}_P$ we define a neighborhood set $Q_{P,q}(4)$, and run the algorithm in Lemma 5. Again we return the output with associated maximum $\mathcal{G}_{Q_{P,q}(4)}(\cdot)$ value, which satisfies Theorem 7.

■ **Algorithm 1** Solving System of Polynomial using an Infinite Grid.

input: a point set $P \subset \mathbb{R}^d$, parameter $\epsilon, \rho > 0$

- 1: **for** each $p \in P$ **do**
- 2: insert p into $Q_{P,q}(4)$ for each $q \in B_p(4\sqrt{\log \frac{1}{\epsilon \rho}}) \cap \text{Grid}(2\sqrt{\frac{\log \frac{1}{\epsilon \rho}}{d}})$
- 3: Let \mathbf{G}_P be the $q \in \text{Grid}(2\sqrt{\frac{\log \frac{1}{\epsilon \rho}}{d}})$ such that $Q_{P,q}(4)$ is non empty
- 4: **for** each $q \in \mathbf{G}_P$ **do**
- 5: Let $x^{(q)}$ be the solution to $\text{SysPoly}(P, q, 2, 4)$ by the algorithm in Lemma 5
- 6: **return** $x' = \text{argmax}_{q \in \mathbf{G}_P} \mathcal{G}_{Q_{P,q}(4)}(x^{(q)})$

► **Theorem 7.** Given $0 < \epsilon, \rho < 1/2$ and a point set $P \subset \mathbb{R}^d$ of size n , let $x^* = \text{argmax}_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_P(x)$. If $\bar{\mathcal{G}}_P(x^*) \geq \rho$, we find $x' \in \mathbb{R}^d$ with $\bar{\mathcal{G}}_P(x') \geq \bar{\mathcal{G}}_P(x^*) - \epsilon \rho$ in time

$$O \left(n \cdot \log n \cdot (2\sqrt{2e\pi})^d + n \cdot \left(\log \frac{d}{\epsilon \rho} \right)^{O(d)} \right).$$

Proof. We need to argue that x^* must be contained in some neighborhood $B_q(2\sqrt{\log \frac{1}{\epsilon\rho}})$ for some $q \in \mathbf{G}_P$, and then apply Lemma 5 with $r = 2$ and $r' = 4$. This follows since, by Lemma 4, $x^* \in B_p(\sqrt{\log \frac{1}{\epsilon\rho}}) \subset B_p(\sqrt{\log \frac{1}{\epsilon\rho}})$ for some $p \in P$. Then let $q \in \mathbf{G}_P$ be the closest grid point to that point p ; the distance $\|p - q\| \leq \sqrt{d}\gamma/2 = \sqrt{\log \frac{1}{\epsilon\rho}}$ with $\gamma = 2\sqrt{\log(1/\epsilon\rho)/d}$. Then by triangle inequality $\|q - x^*\| \leq \|q - p\| + \|p - x^*\| \leq 2\sqrt{\log \frac{1}{\epsilon\rho}}$. Now, we conclude that the output of $\text{SysPoly}(P, p, 2, 4)$ satisfies

$$\begin{aligned} \mathcal{G}_P(x') &\geq \mathcal{G}_{Q_{P,q}(4)}(x') = \mathcal{G}_{Q_{P,q}(4)}(x^{(q)}) \geq \mathcal{G}_{Q_{P,q}(4)}(x^*) - |P| \frac{\epsilon\rho}{2} \\ &= \sum_{p \in P} e^{-\|p - x^*\|^2} - \sum_{p \notin Q_{P,q}(4)} e^{-\|p - x^*\|^2} - |P| \frac{\epsilon\rho}{2} \end{aligned}$$

Note that $\|x^* - p\| \geq \|q - p\| - \|q - x^*\| \geq 4\sqrt{\log \frac{1}{\epsilon\rho}} - 2\sqrt{\log \frac{1}{\epsilon\rho}} = 2\sqrt{\log \frac{1}{\epsilon\rho}}$ since $x^* \in B_q(2\sqrt{\log \frac{1}{\epsilon\rho}})$ and $p \notin B_q(4\sqrt{\log \frac{1}{\epsilon\rho}})$.

$$\mathcal{G}_P(x') \geq \mathcal{G}_P(x^*) - |Q_{P,q}(4)|(\epsilon\rho)^4 - |P| \frac{\epsilon\rho}{2} \geq \mathcal{G}_P(x^*) - |P|\epsilon\rho \quad \text{since } \epsilon, \rho < 1/2$$

We now compute the running time. First, to construct $Q_{P,q}(4)$ (for notation convenience, we use Q_q instead), for each $p \in P$, we enumerate all $q \in B_p(4\sqrt{\log \frac{1}{\epsilon\rho}}) \cap \text{Grid}\left(2\sqrt{\frac{\log \frac{1}{\epsilon\rho}}{d}}\right)$ and insert p into Q_q . Since, by considering the volume of high dimensional sphere, there are $O\left(\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} \left(4\sqrt{\log \frac{1}{\epsilon\rho}} / 2\sqrt{\frac{\log \frac{1}{\epsilon\rho}}{d}}\right)^d\right) = O((2\sqrt{2e\pi})^d)$ points in $B_p(4\sqrt{\log \frac{1}{\epsilon\rho}}) \cap \text{Grid}\left(2\sqrt{\frac{\log \frac{1}{\epsilon\rho}}{d}}\right)$ for each $p \in P$, we have $\sum_{q \in \mathbf{G}_P} |Q_q| = O(n(2\sqrt{2e\pi})^d)$ and also there are only $O(n(4\sqrt{2e\pi})^d)$ non empty Q_q . Here, Γ is the gamma function and we use the fact of $\Gamma(x+1) \geq (\frac{x}{e})^x$. It is easy to construct a data structure to insert all p into all of the corresponding Q_q in $O(n(2\sqrt{2e\pi})^d \log(n(2\sqrt{2e\pi})^d)) = O(n(2\sqrt{2e\pi})^d(\log n + d))$. Let $s = 36e^2 \log \frac{d}{\epsilon\rho}$. We now can precompute each polynomial $\prod_{i=1}^d \left(\sum_{j=0}^{s-1} \frac{1}{j!} (-(x_i - p_i)^2)^j\right)$ in $O(d(2s)^d)$ time for each $p \in P$ which takes $O(nd(2s)^d)$ total time to compute all of them. For each $q \in \mathbf{G}_P$, it takes $O(|Q_q|(2s)^d)$ to construct the polynomial and $O(s^{O(d)})$ time to solve the system of polynomial as suggested in Lemma 5. Therefore, the total running time is

$$\begin{aligned} &O\left(n(2\sqrt{2e\pi})^d(\log n + d) + nd(2s)^d + \sum_{q \in \mathbf{G}_P} (|Q_q|(2s)^d + s^{O(d)})\right) \\ &= O\left(n \cdot \log n \cdot (2\sqrt{2e\pi})^d + n \cdot \left(\log \frac{d}{\epsilon\rho}\right)^{O(d)}\right). \quad \blacktriangleleft \end{aligned}$$

To achieve our final result for low dimensionality, we pre-process the input P by constructing, under the assumption that $\max_x \bar{\mathcal{G}}_P(x) \geq \rho$, a $(1 - \epsilon/3)$ -approximation coresets from [55] of size $O(\frac{d}{\epsilon^2} \frac{1}{\rho} (\log \frac{1}{\rho} + \log \frac{1}{\delta}))$. Running Algorithm 1 on this coresets yields Theorem 1.

Proof of Theorem 1. Let $x^{**} = \operatorname{argmax}_{x \in \mathbb{R}^d} \bar{\mathcal{G}}_{P_1}(x)$. We first have $\bar{\mathcal{G}}_{P_1}(x^{**}) \geq \bar{\mathcal{G}}_{P_1}(x^*) \geq (1 - \frac{1}{3}\epsilon)\bar{\mathcal{G}}_P(x^*) = \Omega(\rho)$ for small ϵ . By Theorem 7 and reparameterizing ϵ , we have

$$\begin{aligned}
\bar{\mathcal{G}}_P(x') &\geq \bar{\mathcal{G}}_{P_1}(x') - \frac{1}{3}\epsilon M_{x'} && \text{by the construction of } P_1 \\
&\geq \bar{\mathcal{G}}_{P_1}(x^{**}) - \frac{1}{3}\epsilon\rho - \frac{1}{3}\epsilon M_{x'} && \text{since } \bar{\mathcal{G}}_{P_1}(x^{**}) = \Omega(\rho) \text{ and by Theorem 7} \\
&\geq \bar{\mathcal{G}}_{P_0}(x^*) - \frac{1}{3}\epsilon\rho - \frac{1}{3}\epsilon M_{x'} \\
&\geq (1 - \frac{1}{3}\epsilon)\bar{\mathcal{G}}_P(x^*) - \frac{1}{3}\epsilon\rho - \frac{1}{3}\epsilon M_{x'} && \text{by } \bar{\mathcal{G}}_P(x^*) \geq \rho \text{ and construction of } P_1 \\
&\geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*) && \text{since } M_{x'} \leq \bar{\mathcal{G}}_P(x^*)
\end{aligned}$$

The final running time is $O(nd)$ to read data and construct P_1 plus

$$O\left(n_1 \cdot \log n_1 \cdot (2\sqrt{2e\pi})^d + n_1 \cdot \left(\log \frac{d}{\epsilon\rho}\right)^{O(d)}\right) = O\left(\frac{d}{\epsilon^2\rho} \cdot \log \frac{1}{\rho\delta} \cdot \left(\log \frac{d}{\epsilon\rho}\right)^{O(d)}\right). \blacktriangleleft$$

3 Dimensionality Reduction for KDE Mode Finding

Leveraging Kirschbraun's extension theorem, we prove that compressing $P = \{p_1, \dots, p_n\}$ using a Johnson-Lindenstrauss random projection to $O(\log n \log^2(1/\epsilon\rho)/\epsilon^2)$ dimensions preserves the mode of the KDE with centers in P , to a $(1 - \epsilon)$ factor. Crucially, we then show that it is possible to recover an approximate mode for P from a solution to the low dimensional problem by applying a *single iteration* of the mean-shift algorithm.

In Section 3.1 we combine this result with our low dimensional algorithm from Section 2 and existing coresets results for KDEs (which allow us to eliminate the $\log n$ dependence) to give our final algorithm for high-dimensional mode finding. We first present the dimensionality reduction result in isolation as, like dimensionality reduction strategies for other computational hard problems [15, 35, 8], it could in principle be combined with any other heuristic or approximate mode finding method. For example, we show a practical strategy is to solve the low-dimensional problem using the mean-shift heuristic.

We need one basic definition before outlining our approach in Algorithm 2.

► **Definition 8** ((γ, k, δ) -Johnson-Lindenstrauss Guarantee). *A randomly selected matrix $\Pi \in \mathbb{R}^{m \times d}$ satisfies the (γ, k, δ) -JL Guarantee if, for any k data points $v_1, \dots, v_k \in \mathbb{R}^d$,*

$$\|v_i - v_j\| \leq \|\Pi v_i - \Pi v_j\| \leq (1 + \gamma)\|v_i - v_j\|,$$

for all pairs $i, j \in 1, \dots, k$ simultaneously, with probability $(1 - \delta)$.

Definition 8 is satisfied by many possible constructions. When Π is a properly scaled random Gaussian or sign matrix, it satisfies the (γ, k, δ) -JL guarantee as long as $m = O(\log(k/\delta)/\gamma^2)$ [17, 1]. In this case, Π can be multiplied by a d dimensional vector in $O(md)$ time. For simplicity, we assume such a construction is used in our algorithm. Other constructions, including fast Johnson-Lindenstrauss transforms [3, 4, 31] and sparse random projections [29, 16] satisfy the definition with slightly larger m , but faster multiplication time. Depending on problem parameters, using such constructions may lead to a slightly faster overall runtime.

■ **Algorithm 2** Dimensionality Reduction for KDE mode finding.

input: a set of n points $P \subset \mathbb{R}^d$, parameters $\epsilon, \delta > 0$, ρ such that $\max_x G_P(x) \geq \rho n$
output: a point $x' \in \mathbb{R}^d$ satisfying $G_P(x') \geq (1 - \epsilon) \max_x G_P(x)$ with prob. $1 - \delta$

- 1: Set $\gamma = \frac{\epsilon}{4 \log(4/\epsilon\rho)}$.
- 2: Choose a random matrix $\Pi \in \mathbb{R}^{m \times d}$ satisfying the $(\gamma, n + 1, \delta)$ -JL guarantee (Defn. 8).
- 3: For each $p_i \in P$, compute Πp_i and let ΠP denote the data set $\{\Pi p_1, \dots, \Pi p_n\}$
- 4: Using an algorithm for mode finding in low dimensions (e.g. Algorithm 1) find a point x'' satisfying $\mathcal{G}_{\Pi P}(x'') \geq (1 - \epsilon/2) \max_{x \in \mathbb{R}^m} \mathcal{G}_{\Pi P}(x)$.
- 5: **return** $x' = \frac{\sum_{p \in P} p \cdot e^{-\|x'' - \Pi p\|^2}}{\sum_{p \in P} e^{-\|x'' - \Pi p\|^2}}$

► **Theorem 9.** *With probability $(1 - \delta)$, Algorithm 2 returns an x' satisfying $\mathcal{G}_P(x') \geq (1 - \epsilon) \max_x \mathcal{G}_P(x)$. When implemented with a random Rademacher or Gaussian Π , the algorithm runs in time $O(ndm) + T_{m, (1-\epsilon/2)}$, where $m = O\left(\frac{\log(n/\delta) \log^2(1/\epsilon\rho)}{\epsilon^2}\right)$ and $T_{m, (1-\epsilon)}$ is the time required to compute a $(1 - \epsilon/2)$ approximate mode for an $O(m)$ dimension dataset.*

The runtime claim is immediate, so we focus on proving the correctness of Algorithm 9. The following key lemma is the main structural result, that the mode of our dimensionality reduced problem has approximately the same density as that of the original. Its proof, as we see below, crucially uses Kirschbraun's extension theorem.

► **Lemma 10.** *Suppose Π is a projection satisfying the $(\gamma, n + 1, \delta)$ -JL guarantee, then*

$$(1 - \epsilon/2) \max_{x \in \mathbb{R}^d} \mathcal{G}_P(x) \leq \max_{x \in \mathbb{R}^m} \mathcal{G}_{\Pi P}(x) \leq \max_{x \in \mathbb{R}^d} \mathcal{G}_P(x) \quad (1)$$

Proof. Let $x^* = \operatorname{argmax}_x \mathcal{G}_P(x)$. Since Π was chosen to satisfy the $(\gamma, n + 1, \delta)$ -JL property with $\gamma = \frac{\epsilon}{4 \log(4/\epsilon\rho)}$, we have that, with probability at least $1 - \delta$, for all $y, z \in \{x^*\} \cup P$,

$$\|y - z\|^2 \leq \|\Pi y - \Pi z\|^2 \leq \left(1 + \frac{\epsilon}{4 \log(4/\epsilon\rho)}\right) \|y - z\|^2. \quad (2)$$

The rest of our analysis conditions on this fact being true. We first prove the left side of (1). From (2), we have that $\|\Pi x^* - \Pi p\|^2 \leq (1 + \frac{\epsilon}{4 \log(4/\epsilon\rho)}) \|x^* - p\|^2$ for all $p \in P$. Accordingly,

$$\begin{aligned} \max_{x \in \mathbb{R}^m} \mathcal{G}_{\Pi P}(x) &\geq \mathcal{G}_{\Pi P}(\Pi x^*) = \sum_{p \in P} e^{-\|\Pi x^* - \Pi p\|^2} \geq \sum_{p \in P} e^{-(1 + \frac{\epsilon}{4 \log(4/\epsilon\rho)}) \|x^* - p\|^2} \\ &\geq \sum_{\substack{p \in P \\ \|x^* - p\|^2 < \log(4/\epsilon\rho)}} e^{-\|x^* - p\|^2} e^{-\frac{\epsilon}{4 \log(4/\epsilon\rho)} \|x^* - p\|^2} \geq (1 - \epsilon/4) \sum_{\substack{p \in P \\ \|x^* - p\|^2 < \log(4/\epsilon\rho)}} e^{-\|x^* - p\|^2}. \end{aligned} \quad (3)$$

The last step uses that $e^{-\frac{\epsilon}{4 \log(4/\epsilon\rho)} \|x^* - p\|^2} \geq 1 - \epsilon/4$ when $\|x^* - p\|^2 \leq \log(4/\epsilon\rho)$. Next we have

$$\sum_{\substack{p \in P \\ \|x^* - p\|^2 < \log \frac{4}{\epsilon\rho}}} e^{-\|x^* - p\|^2} \geq \sum_{p \in P} e^{-\|x^* - p\|^2} - \epsilon n \rho = \mathcal{G}_P(x^*) - \epsilon n \rho \geq (1 - \epsilon/4) \mathcal{G}_P(x^*).$$

This statement follows from two facts: 1) If $\|x^* - p\|^2 \geq \log \frac{4}{\epsilon\rho}$ then $e^{-\|x^* - p\|^2} \leq \epsilon\rho/4$ and 2) we assume that $\mathcal{G}_P(x^*) \geq \rho n$. Combining with (3) we conclude that $\mathcal{G}_{\Pi P}(x) \geq (1 - \epsilon/2) \mathcal{G}_P(x^*)$.

We are left to prove the right hand side of (1). To do so, we rely on the classic Kirschbraun extension theorem for Lipschitz functions, which is stated as follows:

► **Theorem 11** (Kirschbraun Theorem [30, 51]). *For any $\mathcal{S} \subset \mathbb{R}^z$, let $f : \mathcal{S} \rightarrow \mathbb{R}^w$ be an L -Lipschitz function: for all $x, y \in \mathcal{S}$, $\|f(x) - f(y)\|_2 \leq L\|x - y\|_2$. Then there always exists some extension $\tilde{f} : \mathbb{R}^z \rightarrow \mathbb{R}^w$ of f to the entirety of \mathbb{R}^z such that:*

1. $\tilde{f}(x) = f(x)$ for all $x \in \mathcal{S}$,
2. \tilde{f} is also L -Lipschitz: for all $x, y \in \mathbb{R}^z$, $\|\tilde{f}(x) - \tilde{f}(y)\|_2 \leq L\|x - y\|_2$.

We will apply this theorem to the function $g : \{\Pi x^*\} \cup \Pi P \rightarrow \{x^*\} \cup P$ with $g(\Pi y) = y$ for any $y \in \{x^*\} \cup P$. By (2), we have that g is 1-Lipschitz. It follows that there is some function $\tilde{g} : \mathbb{R}^m \rightarrow \mathbb{R}^d$ which agrees with g on inputs $\{\Pi x^*\} \cup P$ and satisfies $\|\tilde{g}(s) - \tilde{g}(t)\| \leq \|s - t\|$ for all $s, t \in \mathbb{R}^m$. This fact can be used to establish that, for any $x \in \mathbb{R}^m$, $\mathcal{G}_{\Pi P}(x) \leq \mathcal{G}_P(\tilde{g}(x))$:

$$\mathcal{G}_{\Pi P}(x) = \sum_{p \in P} e^{-\|x - \Pi p\|^2} \leq \sum_{p \in P} e^{-\|\tilde{g}(x) - \tilde{g}(\Pi p)\|^2} = \sum_{p \in P} e^{-\|\tilde{g}(x) - p\|^2} = \mathcal{G}_P(\tilde{g}(x)).$$

It thus follows that $\max_x \mathcal{G}_{\Pi P}(x) \leq \max_x \mathcal{G}_P(x)$, so the right side of (1) is proven. ◀

In proving Lemma 10 we have also proven the following statement.

► **Corollary 12.** *For any $x \in \mathbb{R}^m$, there exists some point $\tilde{g}(x) \in \mathbb{R}^d$ such that, for all $p \in P$, $\|\tilde{g}(x) - p\| \leq \|x - \Pi p\|$.*

We complete the proof of Theorem 9 by showing that, not only does the maximum of $\mathcal{G}_{\Pi P}$ approximate that of \mathcal{G}_P , but an approximate maximizer for $\mathcal{G}_{\Pi P}$ can be used to recover one for \mathcal{G}_P . Algorithm 2 does so on Line 5 by applying a single iteration of the *mean-shift* algorithm, a common heuristic KDE mode finding [9, 10], which repeatedly iterates the equation $x^{(i+1)} = \frac{\sum_{p \in P} p \cdot e^{-\|x^{(i)} - p\|^2}}{\sum_{p \in P} e^{-\|x^{(i)} - p\|^2}}$. While not guaranteed to converge to a point which maximizes \mathcal{G}_P , a useful property of the mean-shift algorithm is that its solution is guaranteed to never decrease in quality on each iteration:

▷ **Claim 13.** Given $y \in \mathbb{R}^d$, let $y' = \frac{\sum_{p \in P} p \cdot e^{-\|y - p\|^2}}{\sum_{p \in P} e^{-\|y - p\|^2}}$, then $\mathcal{G}_P(y') \geq \mathcal{G}_P(y)$.

Proof. We prove this well known fact for completeness. First, observe by rearrangement that $\mathcal{G}_P(y') - \mathcal{G}_P(y) = \sum_{p \in P} \left(e^{-\|y' - p\|^2 + \|y - p\|^2} - 1 \right) e^{-\|y - p\|^2}$. Then, since $e^z \geq 1 + z$ for all z , we have $e^{-\|y' - p\|^2 + \|y - p\|^2} - 1 \geq -\|y' - p\|^2 + \|y - p\|^2 = -(\|y'\|^2 - \|y\|^2 - 2(y' - y)^T p)$.

$$\begin{aligned} \mathcal{G}_P(y') - \mathcal{G}_P(y) &\geq -(\|y'\|^2 - \|y\|^2) \sum_{p \in P} e^{-\|y - p\|^2} + 2(y' - y)^T \sum_{p \in P} p e^{-\|y - p\|^2} \\ &= \mathcal{G}_P(y) (-\|y'\|^2 + \|y\|^2 + 2(y' - y)^T y') = \mathcal{G}_P(y) \|y' - y\|^2 \geq 0. \end{aligned} \quad \blacktriangleleft$$

Proof of Theorem 9. Recall from Corollary 12 that for any x , there is always a $\tilde{g}(x)$ with

$$\|\tilde{g}(x) - p\| \leq \|x - \Pi p\| \tag{4}$$

for all $p \in P$. Suppose this inequality was tight: i.e., suppose that for all $p \in P, x \in \mathbb{R}^m$, $\|\tilde{g}(x) - p\| = \|x - \Pi p\|$. Then letting x'' be as defined in Algorithm 2, we would have that Line 5 sets x' equal to a mean-shift update applied to $\tilde{g}(x'')$. From Claim 13 we would then immediately have that $\mathcal{G}_P(x') \geq \mathcal{G}_P(\tilde{g}(x'')) = \mathcal{G}_{\Pi P}(x'') \geq (1 - \epsilon/2) \max_x \mathcal{G}_{\Pi P}(x) \geq (1 - \epsilon) \max_x \mathcal{G}_P(x)$, which would prove the theorem.

However, since (4) is not tight, we need a more involved argument by lifting to a $d + 1$ -dimensional space. In particular, for each $p \in P$, let $\bar{p} \in \mathbb{R}^{d+1}$ be a vector with its first d entries equal to p and let the final entry be equal to $\sqrt{\|x - \Pi p\|^2 - \|\tilde{g}(x) - p\|^2}$. Additionally,

■ **Algorithm 3** Full algorithm for high dimensional case.

input: a point set $P \in \mathbb{R}^d$, parameter $\epsilon, \rho, \delta > 0$

- 1: Generate $O(\log \frac{1}{\delta})$ random samples $P_0^j \subset P$ of size $n_0 = O(\frac{1}{\epsilon^2 \rho^2})$ (à la Lopaz-Paz et al.)
- 2: **for** $j \leftarrow 1$ to $O(\log \frac{1}{\delta})$ **do**
- 3: Set $\gamma = \frac{\epsilon}{4 \log(4/\epsilon \rho)}$.
- 4: Choose random matrix $\Pi \in \mathbb{R}^{m \times d}$ satisfying $(\gamma, n+1, 1/100)$ -JL guarantee (Defn. 8)
- 5: For each $p_i \in P_0^j$, compute Πp_i and let ΠP_0^j denote the data set $\{\Pi p_1, \dots, \Pi p_n\}$
- 6: Run the algorithm in Phillips and Tai [39] to construct a subset $P_2^j \subset \Pi P_0^j$ of size $n_2 = O(\frac{\sqrt{m}}{\epsilon \rho} \sqrt{\log \frac{1}{\epsilon \rho}}) = O(\frac{1}{\epsilon^2 \rho} \log^2 \frac{1}{\epsilon \rho})$
- 7: Set x'' as the output of Algorithm 1 (Section 2) on P_2^j in dimension m
- 8: Compute new $x' = \frac{\sum_{p \in P_0^j} p \cdot e^{-\|x'' - \Pi p\|^2}}{\sum_{p \in P_0^j} e^{-\|x'' - \Pi p\|^2}}$
- 9: **Return** the best solution from all iterations of Step 8, evaluated on $\bigcup_j P_0^j$

for every point $x \in \mathbb{R}^m$, let $\tilde{g}(x) \in \mathbb{R}^{d+1}$ be a vector with its first d entries equal to $\tilde{g}(x) \in \mathbb{R}^d$ and final entry equal to 0. Clearly, for any $p \in P$,

$$\|\tilde{g}(x) - \tilde{p}\| = \|x - \Pi p\|. \quad (5)$$

For $z \in \mathbb{R}^{d+1}$, let $\bar{\mathcal{G}}_P(z) = \sum_{p \in P} e^{-\|z - \tilde{p}\|^2}$ and let $\tilde{x}' = \frac{\sum_{p \in P} \tilde{p} e^{-\|x'' - \Pi p\|^2}}{\sum_{p \in P} e^{-\|x'' - \Pi p\|^2}}$. It follows from (5) and the argument above that $\bar{\mathcal{G}}_P(\tilde{x}') \geq \bar{\mathcal{G}}_P(\tilde{g}(x')) = \mathcal{G}_{\Pi P}(x'')$. But clearly it also holds that $\mathcal{G}_P(x') \geq \bar{\mathcal{G}}_P(\tilde{x}')$ because, for any $p \in P$, $\|x' - p\| \leq \|\tilde{x}' - \tilde{p}\|$. So we conclude that $\mathcal{G}_P(x') \geq \mathcal{G}_{\Pi P}(x'')$ as desired. Furthermore, recall that x'' is an approximate mode in the projected setting. It satisfies $\mathcal{G}_{\Pi P}(x'') \geq \max_x (1 - \epsilon/2) \mathcal{G}_{\Pi P}(x)$, and from Lemma 10 we have that $\max_x \mathcal{G}_{\Pi P}(x) \geq (1 - \epsilon/2) \max_x \mathcal{G}_P(x)$. Chaining these inequalities gives the desired bound that $\mathcal{G}_P(x') \geq (1 - \epsilon/2)^2 \max_x \mathcal{G}_P(x) \geq (1 - \epsilon) \max_x \mathcal{G}_P(x)$. ◀

3.1 Final Result for High Dimensions

For the high dimensional case, we combine together the techniques of 1) dimensionality reduction, 2) polynomial system solving and 3) coresets by [34] and [39] to obtain an algorithm that is linear in the dimensionality d and exponential only in $\text{poly}(1/\epsilon, \log 1/\rho)$, leading to Theorem 2.

In the regime where ϵ (the relative error) and δ (the probability of failure) are constant, the runtime simplifies to $O\left(\left(n + \frac{1}{\rho^2}\right)d + \left(\log \frac{1}{\rho}\right)^{O(\log^3 \frac{1}{\rho})}\right)$. Note however that if $1/\rho^2 \leq n_0$ dominates n , then we would not have constructed the coresets P_0^j in the first place but used the entire point set instead, and so we can treat the first term as just $O(nd)$. We also recall that $\rho = \bar{\mathcal{G}}_P(x^*) \geq 1/n$, which by substitution gives an upper bound of $O\left(nd + (\log n)^{O(\log^3 n)}\right)$.

Proof of Theorem 2. We first show the approximation guarantee. It suffices to prove that an iteration of the for loop succeeds with constant probability, so we fix a particular j and omit the superscript in P_0 and P_2 . From Lemma 4, $x^* \in B_q\left(\sqrt{\log \frac{1}{\rho}}\right) \subset B_q\left(\sqrt{\log \frac{1}{\epsilon \rho}}\right)$ for some $q \in P_2$. Let x_0^{**} be $\arg \max_{x \in \mathbb{R}^m} \mathcal{G}_{P_2}(x)$. By Lemma 6 with $r = 1$, we have $\bar{\mathcal{G}}_{P_2}(x'') \geq \bar{\mathcal{G}}_{P_2}(x_0^{**}) - \epsilon \rho \geq (1 - \epsilon) \bar{\mathcal{G}}_{P_2}(x_0^{**})$. The coreset result by [39] implies that, both $|\bar{\mathcal{G}}_{\Pi P_0}(x'') - \bar{\mathcal{G}}_{P_2}(x'')| \leq \epsilon \rho$ and $|\bar{\mathcal{G}}_{\Pi P_0}(x^{**}) - \bar{\mathcal{G}}_{P_2}(x^{**})| \leq \epsilon \rho$ which implies $\bar{\mathcal{G}}_{\Pi P_0}(x'') \geq$

$(1 - O(\epsilon))\bar{\mathcal{G}}_{\Pi P_0}(x^{**})$. Now, let x_0^* be $\arg \max_{x \in \mathbb{R}^d} \mathcal{G}_{\Pi P_0}(x)$. By Theorem 9, with constant probability we have $\mathcal{G}_{P_0}(x') \geq (1 - O(\epsilon))\mathcal{G}_{P_0}(x_0^*)$. By [34], a random sample $P_0 \subset P$ of size $n_0 = O(\frac{1}{\epsilon^2 \rho^2})$ is sufficient to have the guarantee of $|\bar{\mathcal{G}}_P(x) - \bar{\mathcal{G}}_{P_0}(x)| \leq \epsilon \rho$ for any $x \in \mathbb{R}^d$. If we combine this inequality and the guarantee of random sampling, we can conclude that $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*)$.

We now analyze the running time. Reading the input and constructing the coresets P_0^j take $O(nd + n_0 \log \frac{1}{\delta})$ time in total. Evaluating all the solutions in Step 9 takes $O(n_0 d \log^2 \frac{1}{\delta})$ time, since there are $O(\log \frac{1}{\delta})$ many candidates evaluated over a coreset of size $O(n_0 \log \frac{1}{\delta})$ in d dimensions. From Theorem 9, the runtime of a single iteration of the loop is $O(n_0 dm) + T_{m, \epsilon/2}$, where $T_{m, \epsilon/2}$ is the runtime of solving the approximate mode finding problem in m dimensions. In our case, $T_{m, \epsilon/2}$ consists of the runtime of the second coreset result as well as Algorithm 1. It takes time $O(n_0 \text{poly}(1/\epsilon \rho))$ to compute the second coreset P_2^j . Then, Theorem 7 implies that Algorithm 1 requires $O(n_2 \log n_2 \cdot (2\sqrt{2e\pi})^m + n_2 \cdot \left(\log \frac{m}{\epsilon \rho}\right)^{O(m)})$.

The single-loop runtime is dominated by the runtime of Algorithm 1. Writing out the runtime of Algorithm 1 gives

$$\begin{aligned} & O \left(n_2 \log n_2 \cdot (2\sqrt{2e\pi})^m + n_2 \cdot \left(\log \frac{m}{\epsilon \rho} \right)^{O(m)} \right) = O \left(n_2 \cdot \left(\log \frac{m}{\epsilon \rho} \right)^{O(m)} \right) \\ & = O \left(\frac{1}{\epsilon^2 \rho} \log^2 \frac{1}{\epsilon \rho} \cdot \left(\log \frac{1}{\epsilon^3 \rho} \log \frac{1}{\epsilon \rho} \log^2 \frac{1}{\epsilon \rho} \right)^{O(\frac{1}{\epsilon^2} \log^3 \frac{1}{\epsilon \rho})} \right) = O \left(\left(\log \frac{1}{\epsilon \rho} \right)^{O(\frac{1}{\epsilon^2} \log^3 \frac{1}{\epsilon \rho})} \right). \end{aligned}$$

Combining with the runtimes for reading the input, coreset construction and evaluating solutions in Step 9, then repeating the loop for $O(\log \frac{1}{\delta})$ times gives the bound in the theorem statement. If $n < n_0 \log \frac{1}{\delta}$, we use the full set P as each P_0^j . \blacktriangleleft

4 KDE mode finding for Two Dimensional Case

In this subsection, we assume that $P \subset \mathbb{R}^2$ and $p = (p_1, p_2)$ for each $p \in P$. We can improve our low-dimensional analysis that used a set of systems of polynomials by about a logarithmic factor using a different approach. This shows how to approximate each Gaussian by a weighted set of rectangles. After sampling by these weights, we can quickly retrieve the point of maximum depth in these rectangles as an approximation of the maximum.

We first define the following notation. We let $s = \frac{\epsilon \rho}{6}$ be a minimal additive error we will allow for the spatial approximation, and then $m = \lceil \frac{1}{s} \rceil$ will be the number of discretizations we will need. A Gaussian has infinite support, but we will only need to consider m such widths defined $r_j = \sqrt{\log \frac{1}{l_j}}$ with $l_j = 1 - \frac{j}{m}$ for $j = 0, 1, \dots, m$.

As a special case we set $r_m = \infty$ (note that this allows $e^{-r_j^2} = l_j$). We can now define a series of axis-parallel rectangles centered at a point $p = (p_1, p_2) \in P$ as $\mathcal{R}_p = \{[p_1 - r_{a_1}, p_1 + r_{a_1}] \times [p_2 - r_{a_2}, p_2 + r_{a_2}] \mid (a_1, a_2) \in \{0, 1, \dots, m-1\}^2\}$. It enumerates all widths r_0, r_1, \dots, r_{m-1} on both directions, so its size is m^2 . Also, let \mathcal{R} be $\cup_{p \in P} \mathcal{R}_p$.

Given any $x \in \mathbb{R}^d$ and any finite collection \mathcal{C} subsets of \mathbb{R}^2 , denote $N(\mathcal{C}, x)$ as the number of $C \in \mathcal{C}$ that $x \in C$, known as the *depth* or *ply* of x . And we can show that the depth, normalized by $1/(nm^2)$, approximates the KDE value $\bar{\mathcal{G}}_P(x)$.

► **Lemma 14.** $\bar{\mathcal{G}}_P(x) \geq \frac{N(\mathcal{R}, x)}{nm^2} \geq \bar{\mathcal{G}}_P(x) - \frac{1}{3}\epsilon \rho$

The main idea is to show that Gaussian kernel can be approximated by a collection of axis-parallel rectangle where m controls precision. Observe that $|\mathcal{R}| = nm^2$. However, $|\mathcal{R}|$ (and therefore m) does not show up in the running time of our algorithm since, we perform the random sampling on \mathcal{R} in the first step of Algorithm 4.

61:14 Finding an Approximate Mode of a Kernel Density Estimate

Proof. For any $p \in P$ and $i \in \{1, 2\}$, let a_i be the integer such that $r_{a_i-1} \leq |p_i - x_i| \leq r_{a_i}$ which implies $e^{-r_{a_i-1}^2} \geq e^{-(p_i - x_i)^2} \geq e^{-r_{a_i}^2} = 1 - \frac{a_i}{m}$. Then, we have

$$e^{-\|p-x\|^2} = e^{-(p_1-x_1)^2-(p_2-x_2)^2} \geq (1 - \frac{a_1}{m})(1 - \frac{a_2}{m}) = \frac{N(\mathcal{R}_p, x)}{m^2}$$

Note that $N(\mathcal{R}, x) = \sum_{i=1}^d N(\mathcal{R}_p, x)$. Now,

$$\mathcal{G}_P(x) = \sum_{p \in P} e^{-\|p-x\|^2} \geq \sum_{p \in P} \frac{N(\mathcal{R}_p, x)}{m^2} = \frac{N(\mathcal{R}, x)}{m^2}$$

On the other hand, let $\Delta_{p,x,i} = e^{-(p_i-x_i)^2} - (1 - \frac{a_i}{m})$ which is larger than 0,

$$\begin{aligned} \frac{N(\mathcal{R}_p, x)}{m^2} &= (1 - \frac{a_1}{m})(1 - \frac{a_2}{m}) = \left(e^{-(p_1-x_1)^2} - \Delta_{p,x,1}\right) \left(e^{-(p_2-x_2)^2} - \Delta_{p,x,2}\right) \\ &= e^{-(p_1-x_1)^2} e^{-(p_2-x_2)^2} - \Delta_{p,x,1} e^{-(p_2-x_2)^2} - \Delta_{p,x,2} e^{-(p_1-x_1)^2} + \Delta_{p,x,1} \Delta_{p,x,2} \\ &\geq e^{-(p_1-x_1)^2} e^{-(p_2-x_2)^2} - \Delta_{p,x,1} e^{-(p_2-x_2)^2} - \Delta_{p,x,2} e^{-(p_1-x_1)^2} \end{aligned}$$

Recall that $e^{-r_{a_i-1}^2} \geq e^{-(p_i-x_i)^2} \geq e^{-r_{a_i}^2}$ which implies $\Delta_{p,x,i} \leq e^{-r_{a_i-1}^2} - e^{-r_{a_i}^2} = s$. The above equation becomes

$$\begin{aligned} \frac{N(\mathcal{R}_p, x)}{m^2} &\geq e^{-(p_1-x_1)^2} e^{-(p_2-x_2)^2} - \Delta_{p,x,1} e^{-(p_2-x_2)^2} - \Delta_{p,x,2} e^{-(p_1-x_1)^2} \\ &\geq e^{-\|p-x\|^2} - 2s \end{aligned}$$

Finally, we have

$$\frac{N(\mathcal{R}, x)}{m^2} = \sum_{p \in P} \frac{N(\mathcal{R}_p, x)}{m^2} \geq \sum_{p \in P} (e^{-\|p-x\|^2} - 2s) = \mathcal{G}_P(x) - \frac{1}{3} \epsilon n \rho. \quad \blacktriangleleft$$

Now consider (X, \mathcal{S}) be a range space with VC dimension ν . Given $\epsilon > 0$ and $\alpha > 0$, we call a subset Z of X a relative (α, ϵ) -approximation for (X, \mathcal{S}) if, for any $\tau \in \mathcal{S}$, $\left| \frac{|X \cap \tau|}{|X|} - \frac{|Z \cap \tau|}{|Z|} \right| \leq \epsilon M$ when $M = \max\{\frac{|X \cap \tau|}{|X|}, \alpha\}$. A random sample of size $O(\frac{1}{\epsilon^2 \alpha} (\nu \log \frac{1}{\alpha} + \log \frac{1}{\delta}))$ is an (α, ϵ) -approximation with probability at least $1 - \delta$ [25]. The range space $(\mathbb{R}^2, \mathcal{B})$ where \mathcal{B} is the set of all axis-parallel box in \mathbb{R}^2 has VC dimension 4. Thus its dual range space $(\mathcal{B}, \mathcal{D})$ where $\mathcal{D} = \{B \in \mathcal{B} \mid x \in B\} \mid x \in \mathbb{R}^2\}$, has VC dimension is $O(1)$.

Given a set \mathcal{B}_0 of λ axis-aligned rectangles in \mathbb{R}^2 , [12] finds a maximal depth point, that maximizes $N(\mathcal{B}_0, x)$, in $O(\lambda \log \lambda)$ time. This leads to Algorithm 4 and Theorem 3.

■ Algorithm 4 Computing Depth.

input: a point set $P \subset \mathbb{R}^2$, parameter $\epsilon, \rho, \delta > 0$

- 1: generate a random subset \mathcal{R}_0 of \mathcal{R} of size $O\left(\frac{1}{\epsilon^2 \rho} (\log \frac{1}{\rho} + \log \frac{1}{\delta})\right)$.
- 2: compute $x' \in \mathbb{R}^2$ such that $x' = \arg \max_{x \in \mathbb{R}^d} N(\mathcal{R}_0, x)$ using the algorithm by [12].
- 3: **return** x'

Proof of Theorem 3. First, by Lemma 14, $\frac{N(\mathcal{R}, x^*)}{|\mathcal{R}|} \geq \bar{\mathcal{G}}_P(x^*) - \frac{1}{3} \epsilon \rho = \Omega(\rho)$. Let M be $\max\{\frac{N(\mathcal{R}, x')}{|\mathcal{R}|}, \rho\}$. We also have $M = \max\{\frac{N(\mathcal{R}, x')}{|\mathcal{R}|}, \rho\} \leq \bar{\mathcal{G}}_P(x^*)$. By Lemma 14 and the construction of \mathcal{R}_0 , we have $\bar{\mathcal{G}}_P(x') \geq \frac{N(\mathcal{R}, x')}{|\mathcal{R}|} \geq \frac{N(\mathcal{R}_0, x')}{|\mathcal{R}_0|} - \frac{1}{3} \epsilon M$. Since x' is the optimal solution, the term $\frac{N(\mathcal{R}_0, x')}{|\mathcal{R}_0|}$ is larger than $\frac{N(\mathcal{R}_0, x^*)}{|\mathcal{R}_0|}$.

$$\begin{aligned}
\bar{\mathcal{G}}_P(x') &\geq \frac{N(\mathcal{R}_0, x^*)}{|\mathcal{R}_0|} - \frac{1}{3}\epsilon M \\
&\geq \frac{(1 - \frac{1}{3}\epsilon)N(\mathcal{R}, x^*)}{|\mathcal{R}|} - \frac{1}{3}\epsilon M && \text{by } \frac{N(\mathcal{R}, x^*)}{|\mathcal{R}|} = \Omega(\rho) \text{ and construction of } \mathcal{R}_0 \\
&\geq (1 - \frac{1}{3}\epsilon)(\bar{\mathcal{G}}_P(x^*) - \frac{1}{3}\epsilon\rho) - \frac{1}{3}\epsilon M && \text{by [12]}
\end{aligned}$$

Finally, by the assumption of $\rho \leq M \leq \bar{\mathcal{G}}_P(x^*)$, we have $\bar{\mathcal{G}}_P(x') \geq (1 - \epsilon)\bar{\mathcal{G}}_P(x^*)$.

To see the running time, note that the size of input $\lambda = O\left(\frac{1}{\epsilon^2\rho}(\log \frac{1}{\rho} + \log \frac{1}{\delta})\right)$ in our context and $O(n)$ time to create a sample. Therefore, the total running time is

$$O\left(n + \frac{1}{\epsilon^2\rho}(\log \frac{1}{\rho} + \log \frac{1}{\delta}) \log\left(\frac{1}{\epsilon\rho} \log \frac{1}{\delta}\right)\right). \quad \blacktriangleleft$$

5 Experiments

In this section, we present two sets of experiments, demonstrating the efficacy of 1) our dimensionality reduction approach and 2) our 2D combinatorial algorithm. We did not have a ground truth, so we took the best run of any algorithm as the optimal (OPT), and present the “Error in %” as $(x - \text{OPT})/\text{OPT}$. The experiments were run in Python on Google Colab instances with GPU.

Dimensionality Reduction. The first experiment shows the speedup attained via dimensionality reduction, while sacrificing little in solution quality. As noted in Section 3, dimensionality reduction can be combined with any algorithm for mode finding; we compare the state-of-the-art mean-shift heuristic (described also in Section 3) with applying mean-shift after reducing the dimensionality in the data. We use a subset of the CelebA images [33]: $n = 20,000$ aligned and cropped face 178×218 pixel images of celebrities. We converted each image to greyscale, and treat as $(d = 38804)$ -dimensional vectors.

Given the KDE, we pick 10 random starting points and run mean-shift starting at each of them until the KDE value improves by less than 0.001. Then we return to the original dimensionality by running a single iteration of mean shift, to get a final value. We output the best solution, and report the total time of all restarts. For each target dimension (20–800), we report 500 trials as separate marks in the plot. Each trial with reduced dimensionality uses a single JL matrix across all restarts.

Figure 1 (Left) shows that, even if we reduce from 38804 dimensions down to 20 dimensions, the solution quality loss is only in the order of 0.1%. For reference, the solution quality is roughly 6550. The runtime savings are significant, from roughly 130 seconds in the original 38804 dimensions, to 8-9 seconds in 20 dimensions.

The theory demands a JL matrix with one-sided error; the random Gaussian matrix should be divided by some factor of $1 - \epsilon$. We did not do so because: (1) for very low target dimensions (say, 20), there is no valid $\epsilon \in (0, 1)$; and (2) even when such ϵ exists, this ϵ is large enough that a division by $1 - \epsilon$ introduces significant bias and worsens the solution.

2D Algorithm. Figure 1 (Right) shows the comparison of our 2D combinatorial algorithm and heuristics for mode finding. It shows both the best heuristic from [54] of evaluating random points, and then also the mean-shift iterative improvement on top of these [9, 10].

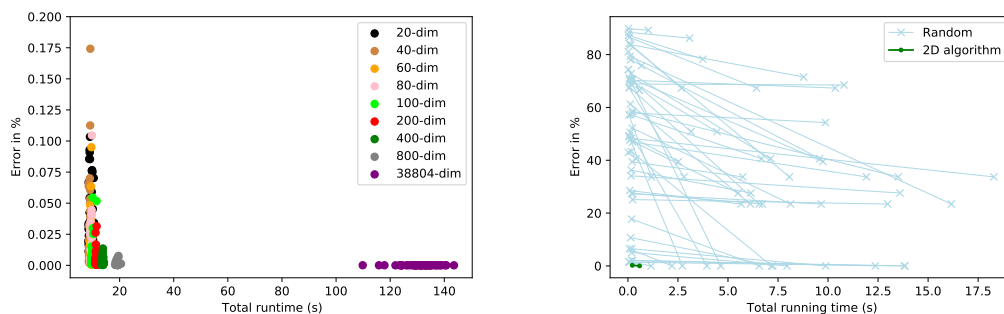


Figure 1 Percentage Error of algorithms as a function of runtime. (Left): Scatterplot for the dimensionality reduction, then mean-shift with 10 restarts. For each target dimension, we show 500 trials. (Right): 2D experiments, our algorithm versus choosing random starting point, then mean-shift. In each segment top point is error before mean-shift, and bottom one is after mean-shift.

We use the entire “Iowa_highway” dataset in [54], which has $n = 1,155,101$ points in \mathbb{R}^2 denoting all waypoints in Iowa from Open Street Maps. It is very multi-modal.

To compare our algorithm, we start with the best heuristic [54] of evaluating the KDE at k data points, and selecting the best. We use k between 1 and 10 random data points, and repeat 5 times for each, and select the lowest error. These are the top blue x s of each segment in the figure. Then for each initial data point, we run mean-shift to improve the error, and report the lowest error (out of each set of k starting points). These are the lower blue x of each segment in the figure. Note that the initial data point sampling heuristic occasionally obtains near-optimal error, but is typically much worse. The blue line segments showing the improvement of mean-shift indicate again it sometimes obtains near-optimal error, but not consistently. It also takes several seconds.

Our algorithm is shown as green dots. The top dot of the green segment is the cost/error of our algorithm, and the lower one is after optimizing with mean shift. We observe that our algorithm is significantly more efficient than the heuristics, taking less than one second, and achieves near-optimality, basically the same error as the best of prior heuristics.

References

- 1 Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.
- 2 Pankaj K Agarwal, Haim Kaplan, and Micha Sharir. Union of random minkowski sums and network vulnerability analysis. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 177–186. ACM, 2013.
- 3 Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, pages 302–322, 2009.
- 4 Nir Ailon and Edo Liberty. An almost optimal unrestricted fast johnson-lindenstrauss transform. *ACM Trans. Algorithms*, 9(3):21:1–21:12, 2013.
- 5 Carlos Améndola, Alexander Engström, and Christian Haase. Maximum number of modes of gaussian mixtures. *Information and Inference: A Journal of the IMA*, 9(3):587–600, 2020.
- 6 Arturs Backurs, Moses Charikar, Piotr Indyk, and Paris Siminelakis. Efficient density evaluation for smooth kernels. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 615–626. IEEE, 2018.

- 7 Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 15799–15808, 2019.
- 8 Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the Johnson-Lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1039–1050, 2019.
- 9 Miguel Á. Carreira-Perpiñán. Mode-finding for mixtures of gaussian distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1318–1323, 2000.
- 10 Miguel Á. Carreira-Perpiñán. Gaussian mean-shift is an em algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):767–776, 2007.
- 11 Miguel A Carreira-Perpinán and Christopher KI Williams. On the number of modes of a gaussian mixture. In *International Conference on Scale-Space Theories in Computer Vision*, pages 625–640. Springer, 2003.
- 12 Timothy M Chan. Klee’s measure problem made easy. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 410–419. IEEE, 2013.
- 13 Cheng Chang and R. Ansari. Kernel particle filter for visual tracking. *IEEE Signal Processing Letters*, 12:242–245, 2005.
- 14 Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman. Robust topological inference: Distance-to-a-measure and kernel distance. Technical report, arXiv:1412.7197, 2014.
- 15 Michael Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *In Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–172, 2015.
- 16 Michael B. Cohen, T.S. Jayram, and Jelani Nelson. Simple analyses of the sparse johnson-lindenstrauss transform. In *The 1st Symposium on Simplicity in Algorithms*, pages 15:1–15:9, 2018.
- 17 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.
- 18 Luc Devroye and László Györfi. *Nonparametric Density Estimation: The L_1 View*. Wiley, 1984.
- 19 Luc Devroye and Gábor Lugosi. *Combinatorial Methods in Density Estimation*. Springer-Verlag, 2001.
- 20 Herbert Edelsbrunner, Brittany Terese Fasy, and Günter Rote. Add isotropic Gaussian kernels at own risk: More and more resilient modes in higher dimensions. *Proceedings 28th Annual Symposium on Computational Geometry*, pages 91–100, 2012.
- 21 Kenji Fukumizu, Le Song, and Arthur Gretton. Kernel bayes’ rule: Bayesian inference with positive definite kernels. *Journal of Machine Learning Research*, 2013.
- 22 Theo Gasser, Peter Hall, and Brett Presnell. Nonparametric estimation of the mode of a distribution of random curves. *Journal of the Royal Statistical Society: Series B*, 60:681–691, 1997.
- 23 Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.
- 24 Mingxuan Han, Michael Matheny, and Jeff M. Phillips. The kernel spatial scan statistic. In *ACM International Conference on Advances in Geographic Information Systems*, 2019.
- 25 Sarel Har-Peled and Micha Sharir. Relative (p, ε) -approximations in geometry. *Discrete & Computational Geometry*, 45(3):462–496, 2011.
- 26 Chi Jin, Yuchen Zhang, Sivaraman Balakrishnan, Martin J. Wainwright, and Michael Jordan. Local maxima in the likelihood of gaussian mixture models: Structural results and algorithmic consequences. In *NeurIPS*, 2016.
- 27 George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, 1995.

- 28 Sarang Joshi, Raj Varma Kommaraji, Jeff M Phillips, and Suresh Venkatasubramanian. Comparing distributions and shapes using the kernel distance. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 47–56. ACM, 2011.
- 29 Daniel M. Kane and Jelani Nelson. Sparser Johnson-Lindenstrauss transforms. *Journal of the ACM*, 61(1):4, 2014.
- 30 M. Kirschbraun. Über die zusammenziehende und lipschitzsche transformationen. *Fundamenta Mathematicae*, 22(1):77–108, 1934.
- 31 Felix Krahmer and Rachel Ward. New and improved johnson-lindenstrauss embeddings via the restricted isometry property. *SIAM Journal on Mathematical Analysis*, 43(3):1269–1281, 2011.
- 32 Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the samples complexity of learning. *Journal of Computer and System Science*, 62:516–527, 2001.
- 33 Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- 34 David Lopaz-Paz, Krikamol Muandet, Bernhard Schölkopf, and Ilya Tolstikhin. Towards a learning theory of cause-effect inference. In *International Conference on Machine Learning*, 2015.
- 35 Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of Johnson-Lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1027–1038, 2019.
- 36 Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10:1–141, 2017.
- 37 Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in euclidean space. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 1064–1069, 2019.
- 38 Jeff M Phillips and Wai Ming Tai. Improved coresets for kernel density estimates. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2718–2727. SIAM, 2018.
- 39 Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 40 Jeff M. Phillips, Bei Wang, and Yan Zheng. Geometric inference on kernel density estimates. In *International Symposium on Computational Geometry*, 2015.
- 41 Kent Quanrud. Spectral sparsification of metrics and kernels. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1445–1464. SIAM, 2021.
- 42 James Renegar. On the computational complexity of approximating solutions for real algebraic formulae. *SIAM Journal on Computing*, 21(6):1008–1025, 1992.
- 43 Alessandro Rinaldo, Larry Wasserman, et al. Generalized density clustering. *The Annals of Statistics*, 38(5):2678–2722, 2010.
- 44 Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.
- 45 David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.
- 46 Chunhua Shen, Michael J. Brooks, and Anton van den Hengel. Fast global kernel density mode seeking: Applications to localization and tracking. *IEEE Transactions on Image Processing*, 16:1457–1469, 2007.
- 47 Bernard W. Silverman. Using kernel density estimates to investigate multimodality. *Journal of the Royal Statistical Society: Series B*, 43:97–99, 1981.
- 48 Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

- 49 Alex J. Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A Hilbert space embedding for distributions. In *Proceedings of Algorithmic Learning Theory*, 2007.
- 50 Bharath K. Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert R. G. Lanckriet. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research*, 11:1517–1561, 2010.
- 51 F. A. Valentine. A lipschitz condition preserving extension for a vector function. *American Journal of Mathematics*, 67(1):83–93, 1945.
- 52 Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theo. of Prob and App*, 16:264–280, 1971.
- 53 Grace Wahba. Support vector machines, reproducing kernel Hilbert spaces, and randomization GACV. In *Advances in Kernel Methods – Support Vector Learning*, pages 69–88. Bernhard Schölkopf and Alezander J. Smola and Christopher J. C. Burges and Rosanna Soentpiet, 1999.
- 54 Yan Zheng and Jeff M Phillips. L_infty error and bandwidth selection for kernel density estimates of large data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1533–1542. ACM, 2015.
- 55 Yan Zheng and Jeff M Phillips. Coresets for kernel regression. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–654. ACM, 2017.
- 56 Shaofeng Zou, Yingbin Liang, H Vincent Poor, and Xinghua Shi. Unsupervised nonparametric anomaly detection: A kernel method. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 836–841. IEEE, 2014.

An Efficient Reduction of a Gammoid to a Partition Matroid

Marilena Leichter ✉

Department of Mathematics, Technische Universität München, Germany

Benjamin Moseley ✉

Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Kirk Pruhs ✉

Computer Science Department, University of Pittsburgh, PA, USA

Abstract

Our main contribution is a polynomial-time algorithm to reduce a k -colorable gammoid to a $(2k - 2)$ -colorable partition matroid. It is known that there are gammoids that can not be reduced to any $(2k - 3)$ -colorable partition matroid, so this result is tight. We then discuss how such a reduction can be used to obtain polynomial-time algorithms with better approximation ratios for various natural problems related to coloring and list coloring the intersection of matroids.

2012 ACM Subject Classification Theory of computation → Network optimization

Keywords and phrases Matroid, Gammoid, Reduction, Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.62

Funding *Marilena Leichter*: Supported in part by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF) and by the Deutsche Forschungsgemeinschaft (DFG), GRK 2201.

Benjamin Moseley: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.

Kirk Pruhs: Supported in part by NSF grants CCF-1535755, CCF-1907673, CCF-2036077 and an IBM Faculty Award.

1 Introduction

Our main contribution is a polynomial-time algorithm to reduce a k -colorable gammoid to a $(2k - 2)$ -colorable partition matroid. Before elaborating on the statement of this result, we first give the necessary definitions, and the most relevant prior work. After stating the result we then explain some of the algorithmic ramifications.

1.1 Definitions

A set system is a pair $M = (S, \mathcal{I})$ where S is a universe of n elements and $\mathcal{I} \subseteq 2^S$ is a collection of subsets of S . Sets in \mathcal{I} are called **independent** and the rank r is the maximum cardinality of a set in \mathcal{I} . A partition C_1, C_2, \dots, C_k of S into independent sets is a k -coloring of M . The **coloring number** of M is the smallest k such that a k -coloring exists.

If each element $s \in S$ has an associated list A_s of allowable colors, then a list coloring is a coloring C_1, C_2, \dots, C_k such that if an $s \in S$ is in C_i then $i \in A_s$. The **list coloring number** of M is the smallest k that guarantees that if for $s \in S$ it is the case that if $|A_s| \geq k$ then a list coloring exists.

If $R \subseteq S$ then the restriction of M to R , denoted by $M \upharpoonright R$, is a set system where the universe is $S \cap R$ and where a set $I \subseteq S$ is independent if and only if $I \subseteq R$ and $I \in \mathcal{I}$.



© Marilena Leichter, Benjamin Moseley, and Kirk Pruhs;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 62; pp. 62:1–62:13

Leibniz International Proceedings in Informatics



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

A hereditary set system is a set system where if $A \subseteq B \subseteq S$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$. A matroid is an hereditary set system with the additional properties that $\emptyset \in \mathcal{I}$ and if $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|A| < |B|$ then there exists an $s \in B \setminus A$ such that $A \cup \{s\} \in \mathcal{I}$. The intersection of matroids $(S, \mathcal{I}_1), \dots, (S, \mathcal{I}_\ell)$ on common universe is a hereditary set system with universe S where a set $I \subseteq S$ is independent if and only if for all $i \in [1, \ell]$ it is the case that $I \in \mathcal{I}_i$.

A **gammoid** is a matroid that has a graphical representation $(D = (V, E), S, Z)$, where $D = (V, E)$ is a directed graph, $S \subseteq V$ is a collection of source vertices and $Z \subseteq V$ is a collection of sink vertices. In the gammoid that is represented by D a set $I \subseteq S$ is in \mathcal{I} if and only if there exists $|I|$ vertex-disjoint paths from the vertices in I to some subcollection of vertices in Z . A **partition matroid** is a type of matroid that can be represented by a partition \mathcal{X} of S . In the partition matroid that is represented by the partition \mathcal{X} a set $Y \subseteq S$ is in \mathcal{I} if and only if $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$.¹

A matroid N is a reduction (also called weak map) of a matroid M , with the same universe, if and only if every independent set in N is also an independent set in M . If N is a partition matroid, then we say that there exists a **partition reduction** from M to N . [5] defined the following decomposability concept, which generalizes partition reduction.

► **Definition 1.** A matroid $M = (S, \mathcal{I})$ is (b, c) -**decomposable** if S can be partitioned into sets X_1, X_2, \dots, X_ℓ such that:

- For all $i \in [\ell]$, it is the case that $|X_i| \leq c \cdot k$, where k is the coloring number of M .
- For a set $Y = \{v_1, \dots, v_\ell\}$, consisting of one representative element v_i from each X_i , the matroid $M \upharpoonright Y$ is b colorable.

If $b = 1$ then X_1, X_2, \dots, X_ℓ represents a partition matroid. Thus $(1, c)$ -decomposability means there exists a partition reduction where the coloring number increases by at most a factor of c .

1.2 Prior Work

There are two prior, independent, papers in the literature that are directly relevant to our results. [3] showed that any gammoid M admits a $(1, (2 - \frac{2}{k}))$ -decomposition. This proof is constructive, and can be converted into an algorithm. The resulting algorithm is essentially a local search algorithm that selects a neighboring solution in the dual matroid in such a way that an auxiliary potential function always decreases. But there seems to be little hope of getting a better than exponential bound on the time, at least using techniques from [3] as the potential can be exponentially large. Further [3] shows that no better bound is achievable.

[5] gave a polynomial-time algorithm to construct a $(18, 1)$ -decomposition of a gammoid. The reduction was shown by leveraging prior work on unsplittable flows [6]. Both paper [3, 5] also observed that partition reductions are relatively easily obtainable for other common types of combinatorial matroids. In particular transversal matroids are $(1, 1)$ -decomposable [3, 5], graphic matroids are $(1, 2)$ -decomposable [3, 5] and paving matroids are $(1, \lceil \frac{r}{r-1} \rceil)$ -decomposable if they are of rank r [3].

The main algorithmic result from [5] is:

► **Theorem 2 ([5]).** Consider matroids M_1, M_2, \dots, M_ℓ defined over a common universe, where matroid M_i has coloring number k_i . There is a polynomial-time algorithm that, given a (b_i, c_i) -decomposition of each matroid M_i , computes a coloring of the intersection of M_1, M_2, \dots, M_ℓ using at most $\left(\prod_{i \in [k]} b_i\right) \cdot \left(\sum_{i \in [k]} c_i\right) k^*$ colors, where $k^* = \max_{i \in [\ell]} k_i$.

¹ Technically one can generalize this to let there be a separate upper bound for each X_i on the number of elements Y can obtain from X_i , but in this paper we only consider partition matroids where the bound is one.

Combining Theorem 2 with the decompositions in [3, 5] one obtains $O(1)$ -approximation algorithms for problems that can be expressed as coloring problems on the intersection of $O(1)$ common combinatorial matroids. Several natural examples of such problems are given in [5]. [1] showed that for two matroids M_1 and M_2 over a common universe with coloring numbers k_1 and k_2 , the coloring number k of $M_1 \cap M_2$ is at most $2 \max(k_1, k_2)$. The proof in [1] uses topological arguments that do not directly give an algorithm for finding the coloring. The list coloring number of a single matroid equals its coloring number [9]. For the intersection of two matroids [3] observed that a list coloring could be efficiently computed by partitioning each of the matroids. A consequence of the results in [3] is a constructive proof that the list coloring number of $M_1 \cap M_2$ is at most $2 \max(k_1, k_2)$ if M_1 and M_2 are each one of the standard combinatorial matroids. Further a consequence of the results in [5] is an efficient algorithm to compute such a list coloring if M_1 and M_2 are each one of the standard combinatorial matroids besides a gammoid.

For hereditary set systems the coloring number is equal to the set cover number. Set cover has been studied extensively in the field of approximation algorithms. The greedy algorithm has an approximation ratio of $H_n \approx \ln n$ and this is essentially optimal assuming $P \neq NP$ [10, 11].

1.3 Our Main Result and Its Algorithmic Applications

We are now ready to state our main result:

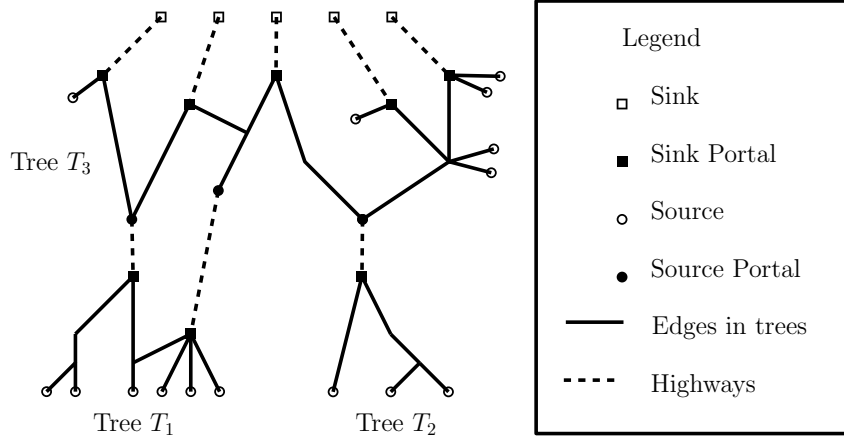
► **Theorem 3.** *A partition reduction from a k -colorable gammoid to a $(2k - 2)$ -colorable partition matroid can be computed in polynomial time given a directed graph D that represents M as input.*

Recall that [3] showed that the $(2k - 2)$ bound is tight.

Combining our main result, Theorem 3, with Theorem 2 from [5] we obtain significantly better approximation guarantees for matroid intersection coloring problems in which one of the matroids is a gammoid. One example is given by the following problem. Initially assume that the input consists of a directed graph D with a designated file server location (a sink) and a collection of clients requesting files from the server at various locations in the networks (the sources). The goal is to as quickly as possible get every client the file that they want from the server, where in each time step one can service any collection of clients for which there exist disjoint paths to the server. This is a matroid coloring problem that can be solved exactly in polynomial-time [4]. Now assume that additionally the input identifies the company to which each client is employed by, and for each company there is a Service Level Agreement (SLA) that upper bounds on how many clients from that company can be serviced in one time unit. Now, the problem becomes a matroid intersection coloring problem, where the intersecting matroids are a gammoid and a partition matroid. Using the $(18, 1)$ -decomposition of a gammoid and Theorem 2 from [5] one obtains a polynomial-time 36-approximation algorithm. However, combining the $(2k - 2)$ -partition reduction of a gammoid from Theorem 3 with Theorem 2 from [5] we now obtain a polynomial-time 3-approximation algorithm.²

Another algorithmic consequence is an efficient algorithm to list coloring the intersection $M_1 \cap M_2$ of a k_1 -colorable matroid M_1 and a k_2 -colorable matroid M_2 if the list of allowable colors for each element has cardinality at least $2 \max(k_1, k_2)$, and each of the matroids is

² This is because Theorem 2 is a $(1, 2)$ -decomposition of a gammoid and a partition matroid is in itself a $(1, 1)$ -decomposition, so Theorem [5] states the approximation is 3.



■ **Figure 1** Example of trees created. Here $k = 3$. Source portals are matched to sink portals along a path not in the trees. All sink portals will have k units of flow entering them and source portals have k units leaving.

either a graphic matroid, paving matroid, transversal matroid, or gammoid. Casting this into the context our running file server example implies that additionally each client has a list of allowable times when the file transfer may be scheduled. Our partition reduction of a gammoid then yields an efficient algorithm to find a feasible schedule as long as the cardinality of allowable times for each client is at least $2 \max(k_1, k_2)$, where k_1 is time required if the network had infinite capacity (so only the SLA constraints come into play), and k_2 is the time required if the SLA allowed infinitely many file transfers (so only the network capacity constraints come into play).

Set cover is a canonical algorithmic problem. So there is considerable interest in discovering examples of natural special types of set cover instances where $o(\log n)$ approximation is possible. For example, several geometrically based types of instances are known, for example covering points in the plane using a discs [7], where a polynomial time approximation scheme is known. Our results provide another example of such a natural special case, namely when the sets come from the intersection of a small number of standard combinatorial matroids.

Theorem 3 and its proof reveal structural properties of gammoids that would seem likely to be of use to address future research on gammoids.

1.4 Overview of Techniques

Given a graphic representation of a gammoid, an optimal coloring can be computed in polynomial-time [4]. By superimposing the source-sink paths for the various color classes one can obtain a flow f from the sources to the sinks that moves at most k units of flow over any vertex. Using standard cycle-canceling techniques [2] one can then convert f to what we call an acyclic flow. A flow f is acyclic if for every undirected cycle C in D at least one edge in C either has flow k or has no flow. Thus by deleting edges that support no flow in f , as they are unnecessary, we are left with a forest \mathcal{T} of edges that have flow in the range $[1, k - 1]$ and a collection of disjoint paths, which we call highways, that have flow k . See Figure 1.

Now each part X in the computed partition \mathcal{X} will be entirely contained in one tree $T \in \mathcal{T}$, and the parts X in a tree $T \in \mathcal{T}$ are computed independently of other trees in \mathcal{T} . There can be four types of vertices in T : (1) sources s that have outflow 1, (2) source portals

\tilde{s} , which are vertices that have a highway directed into them, and which have outflow k in T , (3) sink portals \tilde{z} , which are vertices that have a highway directed out of them, and which have inflow k in T , and (4) normal vertices. Again see Figure 1.

We give a recursive partitioning algorithm for forming the parts X in a tree $T \in \mathcal{T}$. On each recursive step our algorithm first identifies a single part X of at most $2k - 2$ sources and an associated sink portal that are in some sense near each other on the edge of T . The algorithm then removes these sources and sink portal from T , and reconnects disconnected sources back into appropriate places in T . The algorithm then recurses on this new tree T .

Most of the proof that our partitioning algorithm produces a $(1, 2 - \frac{2}{k})$ -decomposition focuses on routing individual trees in \mathcal{T} . So let Y be a collection of sources such that for all $X \in \mathcal{X}$ $|Y \cap X| \leq 1$.

The first key part is proving that as the partitioning algorithm recurses on a tree T , it is always possible to route both the flow coming into T , and the flow emanating within T , out of T , without routing more than k units of flow through any vertex in T . Note that as the algorithm recurses the tree T loses a sink portal (which reduces the capacity of the flow that can leave T by k) and loses up to $2k - 2$ sources (which means there is less flow emanating in T that has to be routed out).

The second key part is to prove that there is a vertex-disjoint routing from the source portals in T and the sources in $T \cap Y$ to the sink portals in T . To accomplish we trace our partitioning algorithm's recursion backwards. So in each step a new collection X of sources and a sink portal is added back into T . We then prove by induction that no matter how the previously considered sources in Y were routed, there is always a feasible way to route the chosen source in $Y \cap X$ to a sink portal in T . Then we finish by observing that unioning the routings constructed within the trees with the highways gives a feasible routing for Y .

2 Preliminaries

This section introduces notation and other necessary definitions. Let $D = (V, E)$ be a directed graph that represents a gammoid. Let $S \subseteq V$ be a set of sources, and $Z \subseteq V$ be the collection of sinks. We may assume without loss of generality that:

- Each vertex $v \in V$ has either out-degree 1 or in-degree 1.
- Each source $s \in S$ has in-degree 0 and out-degree 1.
- Each sink $z \in Z$ has in-degree 1 and out-degree 0 and $|Z| = r$.
- If uv is an edge in E then vu is not an edge in E .

We assume without loss of generality that all color classes have full rank, that is $|S| = rk$. This can be assumed by adding dummy sources to S .

- **Definition 4.** ■ A **feasible flow** in a digraph D from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) p^s is a simple path from s to some sink, and (2) no vertex or edge in D has more than k such paths passing through it.
- A **feasible routing** in a digraph D from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) p^s is a simple path from s to some sink, and (2) no vertex or edge in D has more than one such path passing through it.

3 The Partition Reduction Algorithm

This section gives the Partition Reduction Algorithm. First, we define a corresponding flow graph. Using a Cycle-Canceling Algorithm, we decompose the flow graph into a collection of trees. Then we algorithmically create the partitions from the local structure in these trees. The analysis of the algorithm is deferred to the next section.

3.1 Defining the Flow Graph

Given the digraph D we can compute a minimum k such that M is k -colorable in polynomial time using a polynomial-time algorithm for matroid intersection [8]. Further we can compute the collection of resulting color classes $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$. So \mathcal{C} is a partition of the sources S , and for each $C_i \in \mathcal{C}$ there exist r vertex-disjoint paths p_i^1, \dots, p_i^r in the digraph D from the r sources C_i to Z . We create an f where the flow $f(u, v)$ on each edge (u, v) is initialized to the number of paths p_i^j that traverse (u, v) , that is

$$f(u, v) = \sum_{i=1}^k \sum_{j=1}^r \mathbb{1}_{(u,v) \in p_i^j}$$

A flow f is acyclic if for every undirected cycle C in D at least one edge in C either has flow k or has no flow in f . An arbitrary flow can be converted acyclic by finding cycles in a residual network D^r . This is standard [2], but for completeness we define it here.

For every directed edge (u, v) with $f(u, v) < k$ there exists a forward directed edge (u, v) in D^r with capacity $c_r(u, v) := k - f(u, v)$. For every directed edge (u, v) with $f(u, v) > 0$ there exists a backward directed edge (v, u) in D^r with capacity $c_r(v, u) := f(u, v)$. An augmenting cycle in D^r is a simple directed cycle with strictly more than two edges.

Cycle-Canceling Algorithm. While there exists an augmenting cycle C do the following:

- Let $c := \min_{(u,v) \in C} c_r(u, v)$ be the minimum capacity of an edge in C .
- For each forward edge $(u, v) \in C$, increase $f(u, v)$ by c .
- For each backward edge $(u, v) \in C$, decrease $f(u, v)$ by c .

As every iteration increases the number of edges that have flow k in f or that have no flow in f by 1, the Cycle-Canceling Algorithm terminates after at most $|E|$ iterations. The following observations are straight-forward.

► **Observation 5.** *The following properties hold when the Cycle-Canceling Algorithm terminates:*

- f is a feasible flow of kr units of flow from all the sources.
- Every undirected cycle C in D contains at least one edge with flow k in f or one edge with no flow in f .
- The collection of edges in D that has flow strictly between 0 and k in f forms a forest.
- The collection of edges in D with flow k in f are a disjoint union of directed paths, which we will call **highways**.

3.2 Properties of the Acyclic Flows

We now give several definitions and straightforward observations about our acyclic flow f that will be useful in our algorithm design and analysis.

► **Definition 6.**

- A vertex v is a *source portal* if its in-degree in D is 1, and it has k units of flow passing through it in f .
- A vertex v is a *sink portal* if its out-degree in D is 1, and it has k units of flow passing through it in f .
- Let \mathcal{T} be the forest consisting of edges in D that have flow in f strictly between 0 and k .
- For a tree $T \in \mathcal{T}$ and a vertex $v \in T$ define T_v to be the forest that results from deleting the vertex v from T .

► **Observation 7.** *Each sink $z \in Z$ is in a tree $T \in \mathcal{T}$ that consists solely of z .*

Proof. By assumption, the sink z has in-degree 1 in D and all color classes \mathcal{C} have full rank. Hence, k units of flow are entering z through a unique edge. ◀

As our partition reduction algorithm partitions each tree $T \in \mathcal{T}$ independently, it will be notationally more convenient to fix an arbitrary tree $T \in \mathcal{T}$, and make some definitions relative to this fixed T , and make some observations that must hold for any such T . To a large extent these observations are intended to show that the Figure 2 is accurate.

► **Definition 8.**

- Let \tilde{S} be the collection of source portals in tree T .
- Let \tilde{Z} be the collection of sink portals in tree T .
- A *normal vertex* is a vertex that is none of a source, a sink, a source portal, nor a sink portal.

► **Definition 9.**

- A **feasible flow** in T from a collection $S' \subset S$ is a collection of paths, one path p^s for each $s \in S'$ and k paths $p_1^{\tilde{s}}, \dots, p_k^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) p^s is a simple path from s to some sink portal, (2) each $p_i^{\tilde{s}}$ is a simple path from \tilde{s} to a sink portal, and (3) no vertex or edge in T has more than k such paths passing through it.
- A **feasible routing** in T from a collection $S' \subset S$ is a collection of paths, one path p^s for each $s \in S'$ and one path $p^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) p^s is a simple path from s to some sink portal, (2) $p^{\tilde{s}}$ is a simple path from \tilde{s} to a sink portal, and (3) no vertex or edge in T has more than one such paths passing through it.

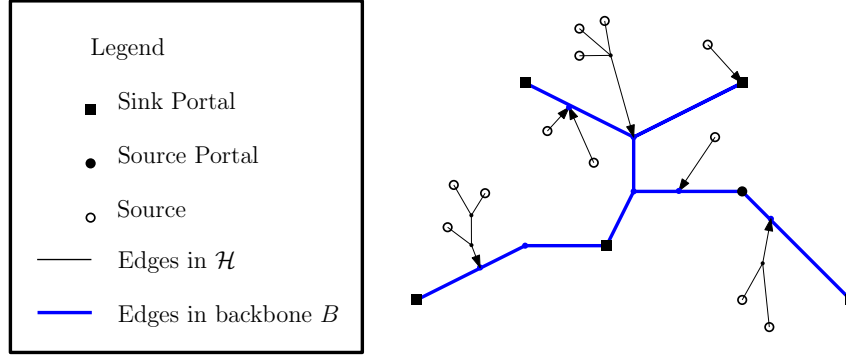
This following observation holds for trees in \mathcal{T} initially and gives intuition for the structure of \mathcal{T} . We remark that this observation may not hold throughout the execution of our algorithm for all trees.

► **Observation 10.** *The number of sources in T is an integer multiple of k .*

Proof. This follows from the fact that each source portal $\tilde{s} \in T$ has exactly k units of flow coming into T via \tilde{s} in the flow f and each sink portal $\tilde{z} \in T$ has exactly k units of flow leaving T via \tilde{z} in f . ◀

► **Definition 11.**

- For two vertices $u, v \in T$, let $P(u, v)$ be the unique undirected path from u to v in T .
- The **backbone** B of T is the subgraph of T consisting of the union of all paths in between pairs of sink portals in T , that is $B = \bigcup_{\tilde{y} \in \tilde{Z}} \bigcup_{\tilde{z} \in \tilde{Z}} P(\tilde{y}, \tilde{z})$.
- For the backbone B , let B_v be the induced forest that results from deleting v from B .
- A vertex v in a backbone B is a **branching vertex** if either:
 - v is not a sink portal and the forest B_v contains at least two trees that each contain exactly one sink portal, or
 - v is a sink portal and the forest B_v contains at least one tree that contains exactly one sink portal.
- Let \mathcal{H} be the forest that results from deleting the edges in B from the tree T .
- For two vertices $u, v \in B$, let $S(P(u, v))$ be the sources $s \in S$ such that there exists a tree $H \in \mathcal{H}$ such that $s \in H$ and such that H contains a vertex $w \in P(u, v)$. Intuitively these are the sources in trees in \mathcal{H} hanging off vertices of the path $P(u, v)$.
- Let $S(v)$ denote $S(P(v, v))$.



■ **Figure 2** Backbone of a tree.

► **Observation 12.** *If $\tilde{s} \in \tilde{S}$ is a source portal in T then \tilde{s} is in the backbone B and $\deg_B^+(\tilde{s}) \geq 2$, that is \tilde{s} has out-degree at least 2 in B .*

Proof. By definition \tilde{s} has a unique incoming edge, which is saturated in f , at least one outgoing edge in T that is not saturated in f . Hence, $\deg_B^+(\tilde{s}) \geq 2$. By flow conservation, there has to be at least two directed paths from \tilde{s} to two different sink portals in T . This implies that \tilde{s} is in the backbone B . ◀

► **Observation 13.** *If B contains at least two sink portals, then B contains a branching vertex v .*

Proof. Consider an arbitrary vertex $v \in B$. If v is not a branching vertex, then there must be a subtree $T' \in B_v$ that contains two sink portals. One can then recurse on T' to find a branching vertex. ◀

► **Observation 14.** *If $s \in S$ is a source in T then s is not in the backbone B .*

Proof. As s has out-degree 1 in D , it can not be on any path between sink portals in T . ◀

► **Observation 15.** *For each tree $H \in \mathcal{H}$ it must be the case that all edges in H are directed towards the unique vertex w in H that is also in B .*

Proof. This follows from the fact that $H \setminus \{w\}$ can not contain a sink portal. ◀

► **Observation 16.** *Assume that T has at least two sink portals. Let v be a branching vertex. Let T' be a tree in the forest B_v that contains exactly one sink portal \tilde{z} . Then the following must hold:*

- $T' = P(v, \tilde{z}) \setminus \{v\}$.
- If T' contains a source portal \tilde{s} , then $\deg_B^+(\tilde{s}) = 2$.
- The path T' contains at most one vertex y such that $\deg_B^+(y) = 2$.

Proof. The first statement follows from the definition of B and the fact that T' only contains one sink portal. The second statement follows since every vertex on a path other than its endpoints has degree two. For the last statement assume to reach a contradiction that there were two such y 's, y_1 and y_2 with y_1 being closer to v in B . Then the flow leaving y_1 toward \tilde{z} could not be feasibly routed through y_2 . ◀

3.3 Description of the Partition Reduction Algorithm

Given the collection of trees \mathcal{T} , our Partition Reduction Algorithm returns a partition \mathcal{X} of the sources in S . The algorithm iterates through the trees T in \mathcal{T} and partitions the sources in T based on their locality in T . So let us consider a particular tree $T \in \mathcal{T}$.

The algorithm performs the first listed case below that applies, with the base cases being checked before the other cases. In the non-base cases the tree T will be modified, and the algorithm called tail-recursively on the modified tree. We will show after the algorithm description that the algorithm maintains the invariant that there is a feasible flow on the tree T throughout the recursion.

Base Case A. If T contains no sources then the recursion terminates, and the algorithm moves to the next tree in \mathcal{T} .

Base Case B. Otherwise if T contains at most $2k - 2$ sources and no source portal then these sources are added as a part X in \mathcal{X} . The recursion terminates, and the algorithm then moves to the next tree in \mathcal{T} .

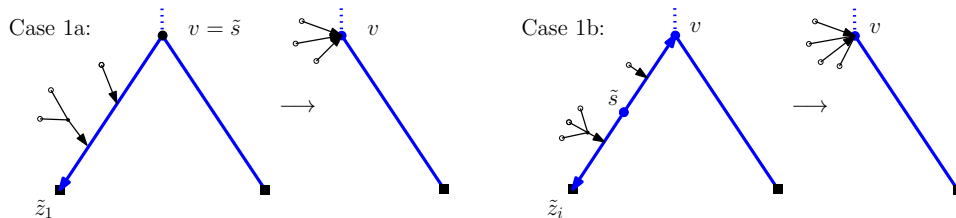
We perform the following recursively on T if neither base case holds. Let v be an arbitrary branching vertex in B . We will show this must exist in Observation 17.

Let \tilde{z}_1 be a sink portal in some tree T_1 in T_v that only contains one sink portal. If v is not a sink portal, let \tilde{z}_2 be a sink portal in some tree T_2 , where $T_1 \neq T_2$, in T_v that only contains one sink portal. If v is a sink portal let $\tilde{z}_2 = v$.

The algorithm's cases are broken up as follows. Case 1 is executed when there is a source portal at v or in T_1 or in T_2 . Case 2 is executed when there is a vertex of out-degree 2 in T_1 or T_2 and there is no source portal. Case 3 is everything else.

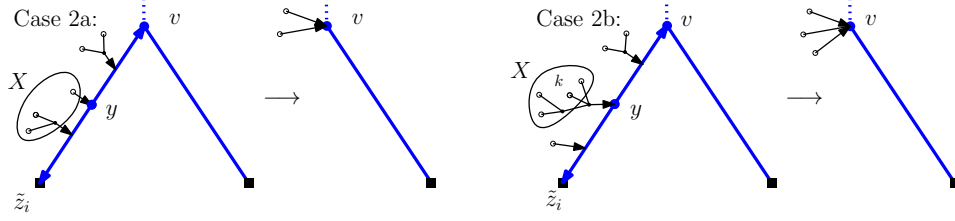
Recursive Case 1a. The vertex v is a source portal. In this case T is modified as follows: (1) for each source $s \in T_1$ a directed edge (s, v) is added to T , (2) v is converted into a normal vertex, and (3) all the nonsources in T_1 are deleted from T . The algorithm then recurses on this new T .

Recursive Case 1b. In this case for some $i \in \{1, 2\}$ the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a source portal. In this case T is modified as follows: (1) for each source $s \in T_i$ a directed edge (s, v) is added to T , and (2) all the nonsources in T_i are deleted from T . The algorithm then recurses on this new T .

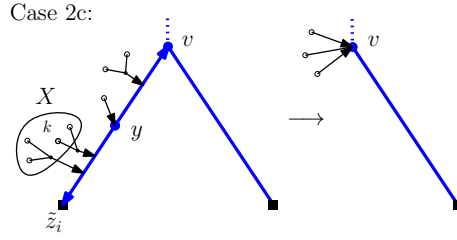


Recursive Case 2a. In this case for some $i \in \{1, 2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg_B^+(y) = 2$ and $|S(P(y, \tilde{z}_i))| \leq 2k - 2$. Add the sources in $S(P(y, \tilde{z}_i))$ as a part X to \mathcal{X} . The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T , and (2) the sources in X and all the nonsources in T_i are deleted from T . The algorithm then recurses on this new T .

Recursive Case 2b. In this case for some $i \in \{1, 2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg_B^+(y) = 2$ and $|S(y)| = k$. In this case the algorithm adds the k sources in $S(y)$ as a part X to \mathcal{X} . The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T , and (2) the sources in X and all the nonsources in T_i are deleted from T . The algorithm then recurses on this new T .

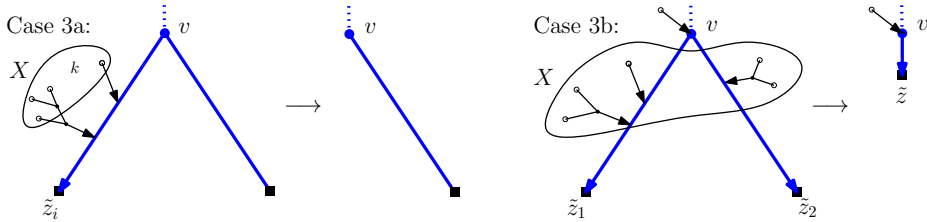


Recursive Case 2c. In this case for some $i \in \{1, 2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg_B^+(y) = 2$ and $|S(P(y, \tilde{z}_i) \setminus \{y\})| = k$. In this case the algorithm adds the k sources in $S(P(y, \tilde{z}_i) \setminus \{y\})$ as a part X to \mathcal{X} . The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T , and (2) the sources in X and all the nonsources in T_i are deleted from T . The algorithm then recurses on this new T .



Recursive Case 3a. In this case for some $i \in \{1, 2\}$ T_i contains exactly k sources. Add the sources in T_i as a part X to \mathcal{X} . The tree T is modified by deleting T_i . The algorithm then recurses on this new T .

Recursive Case 3b. The set of sources in $T_1 \cup T_2$ are added as a part X in \mathcal{X} . The tree T is modified by deleting the vertices in T_1 and T_2 . Add a new sink portal \tilde{z} together with a directed edge (v, \tilde{z}) to T . The algorithm then recurses on this new T .



4 Analysis of the Partition Reduction Algorithm

Our goal is to show that the partition matroid, represented by the partition constructed from the trees, indeed corresponds to a feasible partition reduction from the gammoid. The analysis has the following key components.

- Every tree T has a corresponding feasible flow throughout the algorithm.
- Every part X of the partition has size at most $2k - 2$ and all sources are in some part.
- Any collection of sources Y such that $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ is in \mathcal{I} and, therefore, can each route a unit of flow to the sink in D .

4.1 The Trees Always Have a Feasible Flow

This section's goal is to show that each tree has a feasible flow as defined in Definition 8 throughout the execution of the algorithm. We will later use this to prove that our partition indeed represents a partition matroid that corresponds to a feasible partition reduction in the following section.

We begin by showing various invariants hold for each tree when a feasible flow exists. In particular, this will show that a branching vertex exists if any of the recursive cases are executed. Moreover, arriving at Cases (3a) and (3b) ensure the existence of T_1 and T_2 . All together this with the fact that each tree has a feasible flow will establish that the algorithm always has a case to execute if \mathcal{T} is non-empty.

This observation shows a branching vertex exists if neither base case holds.

► **Observation 17.** *Fix a tree $T \in \mathcal{T}$ during the execution of the algorithm and say T supports a feasible flow as defined in Definition 8. If neither of the base cases apply then T contains at least two sink portals. Moreover a branching vertex must exist in T in this case.*

Proof. This observation holds because T must contain either more than $2k - 2$ sources or a source portal along with at least one source. In either case, we require two sink portals to support the strictly more than k units of flow from these sources and source portal. A branching vertex must then exist by Observation 13. ◀

► **Observation 18.** *Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say T supports a feasible flow as defined in Definition 8. Say that T has a branching vertex v with a tree T_i containing exactly one sink \tilde{z}_i . Moreover say that there is no vertex with out-degree 2 in $P(v, \tilde{z}_i)$. It is the case that $P(v, \tilde{z}_i)$ is a directed path from v to \tilde{z}_i .*

Proof. No vertex with out-degree 2 is in $P(v, \tilde{z}_i)$. Thus, $P(v, \tilde{z}_i)$ is either a path from v to \tilde{z}_i or from \tilde{z}_i to v . Sink portals always have out-degree 0 in T , so the observation follows. We note that, sink portals have out-degree 0 in T initially and are never given outgoing edges by the algorithm. ◀

The next observation shows that a branching vertex is not a sink portal when Cases (3a) or (3b) are executed.

► **Observation 19.** *Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say T supports a feasible flow as defined in Definition 8. Say that T has a branching vertex v and a corresponding tree T_i with exactly one sink \tilde{z}_i . If $P(v, \tilde{z}_i) \setminus \{v\}$ does not contain a vertex of out-degree 2 in B , then v is not a sink portal.*

Proof. Observation 18 implies that $P(v, \tilde{z}_i)$ is a directed path from v to \tilde{z}_i . Sink portals always have out-degree 0 in T , so the observation follows. We note that, sink portals have out-degree 0 in T initially and are never given outgoing edges by the algorithm. ◀

The previous observations guarantee the algorithm always has a case to execute if a feasible flow exists in all trees. The next lemma guarantees the existence of a feasible flow.

► **Lemma 20.** *Fix any tree T during the execution of the algorithm. There must be a feasible flow in T as described in Definition 8.*

4.2 Bounding the Size of the Parts in the Partition

This section shows that every source is in some part X in \mathcal{X} and that every $X \in \mathcal{X}$ has size at most $2k - 2$. Thus we have a valid partition with each part having the desired size.

► **Lemma 21.** *It is the case that $|X| \leq 2k - 2$ for all $X \in \mathcal{X}$. Moreover, every source in S is in some set in \mathcal{X} .*

Proof. It is easy to see that every source in S is in some set in \mathcal{X} . This is because sources are always contained in some tree of \mathcal{T} until they are added to a set placed in \mathcal{X} and the algorithm stops once there is no tree in \mathcal{T} .

Now we show how to bound the size of sets in \mathcal{X} . Cases (1a) and (1b) do not add a set to \mathcal{X} . Cases (2b), (2c) and (3a) add a set to \mathcal{X} of size k by definition. Case (2a) adds a set of size $2k - 2$.

Case (3b) is more interesting. Consider the execution of this case on a tree T with branching vertex v . Let \tilde{z}_1 and \tilde{z}_2 be the corresponding sinks. We know v is not a sink portal by Observation 19 and therefore T_1 and T_2 both exist. By Observation 18, the paths from v to \tilde{z}_1 and \tilde{z}_2 are directed paths from v to \tilde{z}_1 and from v to \tilde{z}_2 . Hence, neither T_1 nor T_2 contains more than k sources. Since case (3a) does not hold, T_1 and T_2 has strictly less than k sources. Thus, there are at most $2k - 2$ sources in the set added to \mathcal{X} . ◀

4.3 Routing Sources within a Tree

In this section, we show that the algorithm is indeed a partition reduction from a k -colorable gammoid to a $(2k - 2)$ -colorable partition matroid. That is, we show that every set Y , that is independent in the partition matroid represented by \mathcal{X} is also independent in the gammoid. More specifically, for any set Y where $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ it is the case that $Y \in \mathcal{I}$ or, equivalently, there is a feasible routing in the digraph D from the sources in Y . The key to this will be Lemma 22 which essentially states that there exists a feasible routing in each tree $T \in \mathcal{T}$.

► **Lemma 22.** *Let T be an arbitrary tree in \mathcal{T} . Let \mathcal{X}_T be the parts in \mathcal{X} that are also in T . For a set Y with $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$, let Y_T be the subset of sources in Y that are also in T . Then there is a feasible routing R in T from Y_T .*

► **Lemma 23.** *Let $Y \subset S$ such that for all $X \in \mathcal{X}$ it is the case that $|X \cap Y| \leq 1$. Then there exists a feasible routing from Y in the digraph D .*

Proof. This follows from Lemma 22 and the fact there is a unique highway into each source portal in each tree and a unique highway leaving each sink portal in every tree. ◀

References

- 1 Ron Aharoni and Eli Berger. The intersection of a matroid and a simplicial complex. *Transactions of the American Math Society*, 2006.
- 2 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.
- 3 Kristóf Bérczi, Tamás Schwarcz, and Yutaro Yamaguchi. List colouring of two matroids through reduction to partition matroids. *arXiv preprint arXiv:1911.10485*, 2019.
- 4 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B:67–72, 1965.
- 5 Sungjin Im, Benjamin Moseley, and Kirk Pruhs. The matroid intersection cover problem. *Operations Research Letters*, 49(1):17–22, 2020.

- 6 Jon M. Kleinberg. Single-source unsplittable flow. In *Symposium on Foundations of Computer Science*, pages 68–77, 1996.
- 7 Nabil Hassan Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 17–22, 2009.
- 8 James Oxley. *Matroid Theory*. Oxford graduate texts in mathematics. Oxford University Press, 2006.
- 9 Paul D Seymour. A note on list arboricity. *Journal of Combinatorial Theory, Series B*, 72(1):150–151, 1998.
- 10 Vijay Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
- 11 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Efficient Algorithms for Least Square Piecewise Polynomial Regression

Daniel Lokshtanov ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

Subhash Suri ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

Jie Xue ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

Abstract

We present approximation and exact algorithms for piecewise regression of univariate and bivariate data using fixed-degree polynomials. Specifically, given a set S of n data points $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ where $d \in \{1, 2\}$, the goal is to segment \mathbf{x}_i 's into some (arbitrary) number of disjoint pieces P_1, \dots, P_k , where each piece P_j is associated with a fixed-degree polynomial $f_j : \mathbb{R}^d \rightarrow \mathbb{R}$, to minimize the total loss function $\lambda k + \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2$, where $\lambda \geq 0$ is a regularization term that penalizes model complexity (number of pieces) and $f : \bigsqcup_{j=1}^k P_j \rightarrow \mathbb{R}$ is the *piecewise polynomial* function defined as $f|_{P_j} = f_j$. The pieces P_1, \dots, P_k are disjoint intervals of \mathbb{R} in the case of univariate data and disjoint axis-aligned rectangles in the case of bivariate data. Our error approximation allows use of any fixed-degree polynomial, not just linear functions.

Our main results are the following. For univariate data, we present a $(1 + \varepsilon)$ -approximation algorithm with time complexity $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$, assuming that data is presented in sorted order of x_i 's. For bivariate data, we present three results: a sub-exponential exact algorithm with running time $n^{O(\sqrt{n})}$; a polynomial-time constant-approximation algorithm; and a quasi-polynomial time approximation scheme (QPTAS). The bivariate case is believed to be NP-hard in the folklore but we could not find a published record in the literature, so in this paper we also present a hardness proof for completeness.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases regression analysis, piecewise polynomial, least square error

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.63

1 Introduction

Line, or curve, fitting is a classical problem in statistical regression and data analysis, where the goal is to find a simple predictive model that best fits an observed data set. For instance, given a set of two-dimensional points (x_i, y_i) , $i = 1, \dots, n$, the least-square line fitting problem is to find a linear function $f : y = ax + b$ minimizing the cumulative error $\sum_{i=1}^n (y_i - (ax_i + b))^2$. This problem is easily solved in $O(n)$ time because the coefficients of the optimal line have a simple closed form solution in terms of input data. In most cases, however, a single line is a poor fit for the data, and instead the goal is to segment the data into multiple piece, with each piece represented by its own linear function. This problem of poly-line (or piecewise linear) fitting has been studied widely in computational geometry, where the goal is either to minimize the total error *for a given number of pieces* [8, 10], or to minimize the number of pieces for a given upper bound on the error [8], under a variety of error measures. In a related but technically different vein of work on “curve simplification”, the approximation must also form a polygonal chain – that is, the pieces representing neighboring segments must form a continuous curve, and it is conjectured that finding a polygonal chain of k pieces with minimum L_2 error is NP-hard [8]. In our regression setting, such continuity is not required.



© Daniel Lokshtanov, Subhash Suri, and Jie Xue;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 63; pp. 63:1–63:15

Leibniz International Proceedings in Informatics



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

These best-fit formulations with a “hard-coded” value for the number of pieces k , however, suffer from the problem of having to specify k , rather than letting the structure in the data dictate the choice. This can be circumvented by running the algorithm for multiple values of k , and then stopping with the smallest number of pieces with an *acceptable* error. A significant issue, however, is the inherent tradeoff between the number of pieces and the error – the larger number of pieces, the smaller the error – which is recognized as the problem of “overfitting” in statistics and machine learning. In order to avoid this overfitting problem, regression typically uses “regularization” and includes a penalty term for the *size* of the representation (model) in the objective, often called the “loss” function. By optimizing the loss function, the algorithm automatically balances the two competing criteria: number of pieces k and approximation error.

In particular, suppose we have a set of data points $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, for $i = 1, \dots, n$. We call (\mathbf{x}_i, y_i) univariate data if $d = 1$ and bivariate if $d = 2$. We will consider piecewise approximation of these data points using polynomial functions of any fixed degree g , where linear functions are the special case when the degree is one. Our goal is to segment \mathbf{x}_i ’s into some (arbitrary) number of disjoint pieces P_1, \dots, P_k , each associated with a constant-degree polynomial function f_j , to minimize the total *loss function*

$$\lambda k + \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2,$$

where $\lambda > 0$ is a pre-specified penalty term for regularizing the model complexity (number of pieces) and $f : \bigsqcup_{j=1}^k P_j \rightarrow \mathbb{R}$ is the *piecewise polynomial* function defined as $f|_{P_j} = f_j$. The pieces P_1, \dots, P_k are disjoint intervals in \mathbb{R} in the case of univariate data and are disjoint axis-aligned rectangles in \mathbb{R}^2 in the case of bivariate data.

Even for piecewise linear approximation of univariate data, the best bound currently known is $\Omega(kn^2)$ [2, 9, 15], and it is an important open problem to either find a sub-quadratic algorithm or prove a $\Omega(n^2)$ lower bound. We make progress on this problem by presenting a linear-time approximation scheme for this problem.

► **Theorem 1.** *There exists a $(1 + \varepsilon)$ -approximation algorithm for univariate piecewise polynomial regression which runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time (excluding the time for pre-sorting).*

For bivariate data, we obtain the following results, including a sub-exponential exact algorithm, a constant-factor approximation in polynomial time, and a quasi-polynomial approximation scheme (QPTAS).

► **Theorem 2.** *There exists an exact algorithm for bivariate piecewise polynomial regression which runs in $n^{O(\sqrt{n})}$ time.*

► **Theorem 3.** *There exists a constant-factor approximation algorithm for bivariate piecewise polynomial regression which runs in polynomial time.*

► **Theorem 4.** *There exists a QPTAS for bivariate piecewise polynomial regression.*

Finally, while the bivariate case (and hence the case of more than two variables) is believed to be NP-hard in the folklore, we could not find a published record in the literature, so we also present a hardness proof for completeness.

► **Theorem 5.** *Bivariate piecewise regression is NP-hard for all fixed degree polynomials, including piecewise constant or piecewise linear functions.*

Related work. Curve fitting and piecewise regression related problems have been studied in computational geometry [6, 8], statistics [16] and machine learning [1, 9] as well as in database theory under the name *histogram approximation* [11, 14]. The main focus of research in computational geometry has been to approximate a curve, or a set of points sampled from a curve, by a fixed-size polygonal chain to minimize some measure of error, such as L_1, L_2, L_∞ error or Hausdorff error. For instance, Goodrich [10] presented an $O(n \log n)$ -time algorithm to compute a polyline (or a connected piecewise linear function) in the plane that minimizes the maximum vertical distance from a set of n points to the polyline, which improves upon the algorithms of [12, 17]. Aronov et al. present an FPTAS for the polyline fitting problem with the min-sum and least-square error measure, and conjecture that finding a polygonal chain of k pieces with minimum L_2 error is NP-hard [8]. Agarwal et al. [6] consider approximation under Hausdorff and Frechet distances.

Unlike these computational geometric models, in statistics, machine learning and database theory, the piecewise approximation is typically *not* required to be “connected”; instead, the goal is to partition the data into a given number k of pieces, each represented by a simple function. Such an optimal histogram (piecewise approximation) can be constructed in $O(kn^2)$ time using dynamic programming, where k is the number of pieces [11, 14]. A similar dynamic programming algorithm can also compute an optimal “regularized” piecewise approximation, where k is the number of pieces in the optimal solution [15]. It is an important open problem to either find a sub-quadratic algorithm or prove a $\Omega(n^2)$ lower bound.

In machine learning, Acharya et al. [2] study a “segmented regression” problem where the goal is to recover a function f , which is promised to be “nice” (say, piecewise linear with k pieces), and the sampled data from f has a small random noise. The quality of recovery is measured by the mean squared error. In this model, they present an algorithm for computing a function with $O(k)$ linear pieces in $O(n \log n)$ time [2]. An extension to multi-dimensional data with similar results is presented in [9]. Our focus is a little different from these results because (1) we do not assume a fixed value of k , and (2) we judge the error of our regression against worst-case input that is not necessarily drawn from a hypothetical k -piece input with small random noise. Thus, these two lines of research are complementary.

Finally, for bivariate data, Agarwal and Suri [7] considered the problem of computing a piecewise linear surface with smallest number of pieces whose vertical distance from data points is at most ε . They showed that the problem is NP-hard and gave a polynomial-time $O(\log n)$ -approximation algorithm.

Organization. Section 2 introduces some basic notations and concepts used throughout the paper. Our linear-time approximation scheme for univariate data (Theorem 1) is presented in Section 3, while our algorithms for bivariate data are presented in Section 4. Finally, in Section 5, we conclude the paper and pose some open questions. Due to limited space, the algorithm of Theorem 2 and the hardness result of Theorem 5 (as well as some proofs and details) are omitted in this version, which will appear in the full paper.

2 Basic notations and concepts

We begin with basic notation and concepts that are used throughout the paper. For an integer $g \geq 0$, we use $\mathbb{R}[x]_g$ and $\mathbb{R}[x, x']_g$ to denote the family of all univariate and bivariate polynomial functions with degree at most g . A univariate (resp., bivariate) *piecewise polynomial function* of degree at most g is a function $f : \bigsqcup_{j=1}^k P_j \rightarrow \mathbb{R}$, where P_1, \dots, P_k are disjoint intervals in \mathbb{R}^1 (disjoint axis-parallel rectangles in \mathbb{R}^2) and $f|_{P_j} = f_j|_{P_j}$ for some

$f_j \in \mathbb{R}[x]_g$ (resp., $f_j \in \mathbb{R}[x, x']_g$), for all $j \in \{1, \dots, k\}$. The intervals (resp., rectangles) P_1, \dots, P_k are the *pieces* of f , and the number k is the *complexity* of f , denoted by $|f|$. The notion of piecewise polynomial functions generalizes to higher dimensions (multi-variables), where the pieces becomes axis-parallel boxes but in this paper we only study univariate and bivariate piecewise polynomial functions.

Let Γ_g^d denote the family of piecewise polynomial functions with d variables and of degree at most g . For a set of n points $S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^n$, we define the *error* of a function $f \in \Gamma_g^d$ for S as

$$\sigma_S(f) = \lambda \cdot |f| + \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2,$$

where $\lambda > 0$ is a pre-specified (regularizer) parameter. We assume $\sigma_S(f) = \infty$ if the domain of f does not cover all \mathbf{x}_i 's. For a fixed constant g , the *piecewise polynomial regression* problem takes S and λ as the input, and aims to find the function $f^* \in \Gamma_g^d$ that minimizes the error $\sigma_S(f^*)$. By appropriate scaling of the y -values in the input, we can assume without loss of generality that $\lambda = 1$. Therefore, for convenience, we make this assumption throughout the paper.

3 Algorithm for univariate data

The input to the univariate regression problem is a dataset $S = \{(x_i, y_i) \in \mathbb{R} \times \mathbb{R}\}_{i=1}^n$, where $x_1 \leq \dots \leq x_n$, and the goal is to find the function $f^* \in \Gamma_g^1$ minimizing $\sigma_S(f^*)$, for some fixed constant $g \geq 0$, where we assume $\lambda = 1$, as mentioned earlier. This problem can be solved in $O(n^2)$ time with a straightforward dynamic program, and no subquadratic-time (even approximation) algorithm is known. Our main result in this section is a linear-time approximation scheme, which for any $\varepsilon > 0$ computes in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time a piecewise function $f \in \Gamma_g^1$ whose error is at most $(1 + \varepsilon) \cdot \text{opt}$, assuming that the points in S are pre-sorted by their x -coordinates.

In order to explain the main ideas behind our algorithm, it is helpful to first briefly review the quadratic-time dynamic programming algorithm. That algorithm performs n iterations, where the i th iteration computes an optimal piecewise regression for the subset of points $(x_1, y_1), \dots, (x_i, y_i)$. If the rightmost piece in the optimal solution for this subproblem covers the points $(x_j, y_j), \dots, (x_i, y_i)$, then the solution combines the optimal regression for $(x_1, y_1), \dots, (x_{j-1}, y_{j-1})$ with the best fitting degree g polynomial for $(x_j, y_j), \dots, (x_i, y_i)$. By dynamic programming, the former is already computed in the $(j-1)$ th iteration, and the latter can be computed for all subproblems with an $O(n^2)$ -time preprocessing step. There are $O(i)$ candidates for the rightmost piece, and so the i th iteration takes $O(i)$ time, resulting in an $O(n^2)$ time algorithm.

A natural idea for improving the dynamic program's time complexity is to reduce the number of guesses needed for the rightmost piece in each iteration: ideally, we would like to find the “best” rightmost piece without trying all possibilities. This, however, seems quite difficult if we want the exact optimal solution. Our main idea is to show that this is possible if we only need a $(1 + \varepsilon)$ approximation of the minimum error. Our algorithm builds on three key steps. First, we prove a structural lemma (Lemma 7) showing that there exists an approximate solution f in which the squared error of each piece (essentially) is bounded by $O(1/\varepsilon)$, and therefore contributes between 1 and $1 + O(1/\varepsilon)$ to the final objective $\sigma_S(f)$. The second key idea is to show that, for each $i \in [n]$, there exist a set of $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ “candidate” pieces with right endpoint x_i such that a $(1 + \varepsilon)$ -approximate solution can be found using

only these pieces (Lemma 8). Thus, assuming that these candidate pieces and their best fit degree g polynomials are known, we only have to make $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ guesses in each iteration, which leads to an $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ -time algorithm. The final, and third, step is to compute all the candidate pieces efficiently, which we show can be done using prefix sum and the standard formula for least-square polynomial regression—the details of this part will appear in the full paper.

With this preamble, we are ready to describe our algorithm in detail. For $a, b \in [n]$ satisfying $a \leq b$, we define

$$f[a, b] = \arg \min_{f \in \mathbb{R}[x]_g} \sum_{i=a}^b (y_i - f(x_i))^2 \quad \text{and} \quad \delta[a, b] = \min_{f \in \mathbb{R}[x]_g} \sum_{i=a}^b (y_i - f(x_i))^2.$$

That is, $f[a, b]$ is the best-fit polynomial in $\mathbb{R}[x]_g$ for the set of points $(x_a, y_a), \dots, (x_b, y_b)$ (in terms of square error) and $\delta[a, b]$ is the square error of $f[a, b]$. We have the following simple observation.

► **Lemma 6.** *If $a' \leq a$ and $b' \geq b$, then $\delta[a', b'] \geq \delta[a, b]$. Furthermore, for a sequence of numbers a_0, a_1, \dots, a_r where $a-1 \leq a_0 < \dots < a_r \leq b$, we have $\delta[a, b] \geq \sum_{j=1}^r \delta[a_{j-1}+1, a_j]$.*

Let ε be the approximation factor, which we assume is sufficiently small, say $0 < \varepsilon \leq 1$. Let $\tilde{\varepsilon} > 0$ be such that it satisfies $(1 + \tilde{\varepsilon})^2 = 1 + \varepsilon$. Then, we have $\varepsilon/3 \leq \tilde{\varepsilon} \leq \varepsilon$ since $\varepsilon \leq 1$. For an index $i \in [n]$, we say i is a *left* (resp., *right*) *break point* if $x_{i-1} < x_i$ (resp., $x_{i+1} > x_i$). For a function $f \in \Gamma_g^1$ and a piece P of f , the *cost* of P is defined as $\sum_{x_i \in P} (y_i - f(x_i))^2$. Thus, the total error $\sigma_S(f)$ is simply $|f|$ plus the cost of all the pieces of f .

► **Lemma 7.** *There exists a function $f \in \Gamma_g^1$ such that $\sigma_S(f) \leq (1 + \tilde{\varepsilon}) \cdot \text{opt}$ and each piece of f either has cost at most $2/\tilde{\varepsilon}$ or is a singleton point.*

Proof. Let $f^* \in \Gamma_g^1$ be an optimal solution, and so $\sigma_S(f^*) = \text{opt}$. Consider a piece of f^* , say, $P^* = [x_a, x_b]$ where a is a left break point and b is a right break point and $a, b \in [n]$. Since f^* is optimal, the cost of P^* is $\delta[a, b]$. We replace P^* with $r < \tilde{\varepsilon} \cdot \delta[a, b] + 1$ pieces P_1, \dots, P_r as follows. We say a pair (a', a'') of indices with $a' \leq a''$ is *legal* if $x_{a'} = x_{a''}$ or $\delta[a', a''] \leq 2/\tilde{\varepsilon}$. Starting with $a_0 = a - 1$, we create a sequence a_0, a_1, a_2, \dots of indices, where a_{i+1} is the largest right break point in $\{a_i + 1, \dots, b\}$ such that $(a_i + 1, a_{i+1})$ is legal. The sequence ends at some $a_r = b$, and we claim that $r < \tilde{\varepsilon} \cdot \delta[a, b] + 1$. We first observe that since a_{i+1} is the largest right break point for which $(a_i + 1, a_{i+1})$ is legal, we have $\delta[a_i + 1, a_{i+2}] > 2/\tilde{\varepsilon}$ for all $i \in \{0, 1, \dots, r-2\}$. Now consider the sum $\sum_{i=0}^{\lfloor r/2 \rfloor - 1} \delta[a_{2i} + 1, a_{2(i+1)}]$. Each summand of this sum is greater than $2/\tilde{\varepsilon}$. On the other hand, we have $\delta[a, b] \geq \sum_{i=0}^{\lfloor r/2 \rfloor - 1} \delta[a_{2i} + 1, a_{2(i+1)}]$ by Lemma 6. It directly follows that $\lfloor r/2 \rfloor < \tilde{\varepsilon} \cdot \delta[a, b]/2$ and hence $r < \tilde{\varepsilon} \cdot \delta[a, b] + 1$. We define $P_i = [x_{a_{i-1}+1}, x_{a_i}]$ for $i \in [r]$. We replace P^* of f^* with P_1, \dots, P_r , and call them the *sub-pieces* of P^* . We do this for all pieces of f^* , which gives us our function $f \in \Gamma_g^1$, as follows. First, clearly, the domain of f is contained in the domain of f^* . Next, for each piece $P = [x_a, x_b]$ of f , the function $f|_P$ is simply the polynomial $f[a, b]$ restricted to P , whose cost is $\delta[a, b]$. All that remains is to bound the total error $\sigma_S(f)$. Consider a piece $P^* = [x_a, x_b]$ of f^* and its sub-pieces P_1, \dots, P_r . Let $c(P^*)$ be the total cost of all the sub-pieces P_1, \dots, P_r plus r . By Lemma 6, the total cost of all the sub-pieces P_1, \dots, P_r is at most $\delta[a, b]$, and since $r < \tilde{\varepsilon} \cdot \delta[a, b] + 1$ and $c^*(P^*) = \delta[a, b] + 1$, we get $c(P^*) \leq (1 + \tilde{\varepsilon}) \cdot c^*(P^*)$. This inequality holds for each piece of f^* , and so we get our result that $\sigma_S(f) \leq (1 + \tilde{\varepsilon}) \cdot \sigma_S(f^*)$. ◀

For convenience, we say a function $f \in \Gamma_g^1$ is *S-light* if each piece of f is either a singleton point or of cost at most $2/\tilde{\varepsilon}$. Similarly, for a subset $S' \subseteq S$, we say a function $f \in \Gamma_g^1$ is *S'-light* if each piece of f is either a singleton point or of cost *with respect to* S' (i.e., the sum of only the square error of the points in S') at most $2/\tilde{\varepsilon}$.

For a right break point $b \in [n]$ and an integer $i \geq 0$, let $a_i(b) \in [b]$ be the smallest left break point such that $\delta[a_i(b), b] \leq (1 + \varepsilon)^i - 1$; if such a left break point does not exist, we set $a_i(b)$ to be the largest left break point that is smaller than or equal to b . We define an index set $A(b) = \{a_i(b) : i \geq 0 \text{ and } (1 + \varepsilon)^{i-1} - 1 \leq 2/\varepsilon\}$. We say an interval I is *canonical* if $I = [x_a, x_b]$ for some $a, b \in [n]$ such that b is a right break point and $a \in A(b)$. A function $f \in \Gamma_g^1$ is *canonical* if all pieces of f are canonical intervals. The following lemma shows that we can limit our search to canonical functions.

► **Lemma 8.** *There exists a canonical function $f \in \Gamma_g^1$ such that $\sigma_S(f) \leq (1 + \varepsilon) \cdot \text{opt}$.*

Proof. We claim that for any S -light function $f_0 \in \Gamma_g^1$, there exists a canonical function $f \in \Gamma_g^1$ with $\sigma_S(f) \leq (1 + \varepsilon) \cdot \sigma_S(f_0)$. This claim in combination with Lemma 7 proves the lemma. We prove the claim using induction on the number r of distinct x -coordinates of the points in S , i.e., distinct elements in $\{x_1, \dots, x_n\}$. If $r = 1$, then $x_1 = \dots = x_n$ and the interval $I = [x_1, x_n]$ is a singleton point. Furthermore, in this case, 1 is the unique left break point, hence $1 \in A(n)$ and I is canonical. Therefore, the claim clearly holds. Assume that the claim holds if the number of distinct x -coordinates of the points in S is less than r , and consider the case where the number is r . Let $f_0 \in \Gamma_g^1$ be a S -light function, and we want to show that there exists a canonical function $f \in \Gamma_g^1$ such that $\sigma_S(f) \leq (1 + \varepsilon) \cdot \sigma_S(f_0)$. Consider the rightmost piece P of f_0 . Without loss of generality, we may assume that $P = [x_a, x_n]$ for some left break point $a \in [n]$. Let $c(P)$ be the cost of P . We consider two cases, $c(P) \leq 2/\varepsilon$ and $c(P) > 2/\varepsilon$. If $c(P) \leq 2/\varepsilon$, we define i as the smallest integer such that $(1 + \varepsilon)^i \geq c(P) + 1$. Therefore, $(1 + \varepsilon)^{i-1} \leq c(P) + 1 \leq (1 + \varepsilon)^i$. Since $c(P) \leq 2/\varepsilon$, we have $(1 + \varepsilon)^{i-1} - 1 \leq 2/\varepsilon$ and hence $a_i(n) \in A(n)$. By the definition of $a_i(n)$, we have $a_i(n) \leq a$ and $\delta[a_i(n), n] \leq (1 + \varepsilon)^i - 1$, i.e., $\delta[a_i(n), n] + 1 \leq (1 + \varepsilon)^i$. Since $(1 + \varepsilon)^{i-1} \leq c(P) + 1$, we further deduce that $\delta[a_i(n), n] + 1 \leq (1 + \varepsilon) \cdot (c(P) + 1)$. Now we define $S' = \{(x_1, y_1), \dots, (x_{a-1}, y_{a-1})\} \subseteq S$ and $S'' = \{(x_1, y_1), \dots, (x_{a_i(n)-1}, y_{a_i(n)-1})\} \subseteq S$. Let $f'_0 \in \Gamma_g^1$ be the function obtained by restricting f_0 to the union of the pieces other than P . Then f'_0 is both S' -light and S'' -light. Note that the number of distinct x -coordinates of the points in S'' is strictly less than r , as $a_i(n)$ is a left break point. Therefore, by our induction hypothesis, there exists some canonical function $f'' \in \Gamma_g^1$ with $\sigma_{S''}(f'') \leq (1 + \varepsilon) \cdot \sigma_{S''}(f'_0) \leq (1 + \varepsilon) \cdot \sigma_{S'}(f'_0)$, and we can assume without loss of generality that all pieces of f'' are contained in the range $(-\infty, x_{a_i(n)-1}]$. We define our function f as the “combination” of f'' and $f[a_i(n), n]$. Specifically, the pieces of f consists of all pieces of f'' and the interval $[x_{a_i(n)}, x_n]$. On the piece $[x_{a_i(n)}, x_n]$, f is the same as $f[a_i(n), n]$. On the other pieces, f is the same as f'' . Clearly, $f \in \Gamma_g^1$, and it is canonical because f'' is canonical and $[x_{a_i(n)}, x_n]$ is a canonical interval. Finally, we have

$$\begin{aligned} \sigma_S(f) &= \sigma_{S''}(f'') + \delta[a_i(n), n] + 1 \\ &\leq (1 + \varepsilon) \cdot \sigma_{S'}(f'_0) + (1 + \varepsilon) \cdot (c(P) + 1) \\ &= (1 + \varepsilon) \cdot \sigma_S(f_0). \end{aligned}$$

In the case $c(P) > 2/\varepsilon$, P must be a singleton point as f_0 is S -light. Thus, $x_a = x_n$ and a is the largest left break point smaller than or equal to n , which implies $a_0(n) = a$ and hence P is canonical. By our induction hypothesis, there exists some canonical function $f'' \in \Gamma_g^1$ with $\sigma_{S'}(f'') \leq (1 + \varepsilon) \cdot \sigma_{S'}(f'_0)$, where $S' = \{(x_1, y_1), \dots, (x_{a-1}, y_{a-1})\}$. Without loss of generality, we may assume all pieces of f'' are contained in the range $(-\infty, x_{a-1}]$. Similarly to the above, We define f as the combination of f'' and $f[a, n]$. Since $\sigma_{S'}(f'') \leq (1 + \varepsilon) \cdot \sigma_{S'}(f'_0)$ and the cost of P is at least $\delta[a, n]$, we have $\sigma_S(f) \leq (1 + \varepsilon) \cdot \sigma_S(f_0)$. ◀

We can find a canonical function $f \in \Gamma_g^1$ minimizing $\sigma_S(f)$ using dynamic programming, as shown in Algorithm 1. By Lemma 8, the result is a $(1 + \varepsilon)$ -approximation of the univariate regression problem.

■ **Algorithm 1** APPROXIMATE-REGRESSION-1D(S).

```

1:  $t \leftarrow 0$  and  $\text{opt}_0 \leftarrow 0$ 
2: for  $t$  from 1 to  $n$  do
3:   if  $t$  is a right break point then
4:      $\tilde{a} \leftarrow \arg \min_{a \in A(t)} \{\text{opt}_{a-1} + (\delta[a, t] + 1)\}$ 
5:      $\text{opt}_t \leftarrow \text{opt}_{\tilde{a}-1} + (\delta[\tilde{a}, t] + 1)$ 
6: return  $\text{opt}_n$ 

```

The correctness of Algorithm 1 is clear. To analyze its time complexity, we observe that $|A(b)| = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ for all right break points $b \in [n]$. Therefore, assuming that we know all the index sets $A(b)$ and all the $f[a, b]$ and $\delta[a, b]$, where $a \in A(b)$, Algorithm 1 can be directly implemented in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time. The details of how to compute all $A(b)$ and all $f[a, b], \delta[a, b]$, where $a \in A(b)$, in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time will appear in the full paper. The following theorem states the main result of this section.

► **Theorem 1.** *There exists a $(1 + \varepsilon)$ -approximation algorithm for univariate piecewise polynomial regression which runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time (excluding the time for pre-sorting).*

4 Algorithms for bivariate data

In this section, we present our algorithms for piecewise polynomial regression for bivariate data. The input of the problem is a dataset $S = \{(x_i, x'_i), y_i\} \in \mathbb{R}^2 \times \mathbb{R}\}_{i=1}^n$, and our goal is to find a function $f^* \in \Gamma_g^2$ that minimizes $\sigma_S(f^*)$ (recall that $\lambda = 1$ by assumption).

We present three algorithms for this problem. The first is a polynomial-time constant-factor approximation. This is the simplest of the three results. The second algorithm computes the exact solution in sub-exponential time $n^{O(\sqrt{n})}$, which makes use of the planar separator theorem (this one will appear in the full paper). The third result is a quasi-polynomial time approximation scheme, and is technically the most sophisticated of the three algorithms.

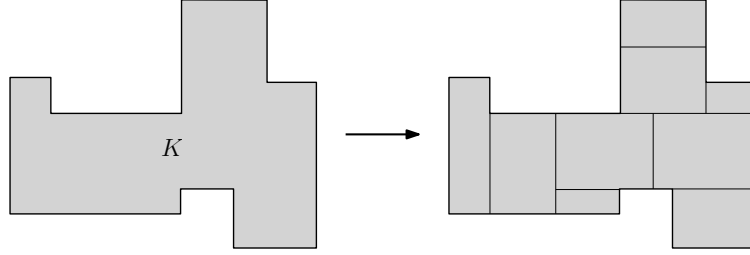
We begin with a brief overview of the high-level ideas underlying our algorithms. We first observe that a piecewise function corresponds to an orthogonal partition of the plane (induced by the pieces of the function). Therefore, the problem of finding the optimal function $f^* \in \Gamma_g^2$ is (essentially) equivalent to computing an optimal orthogonal partition of the plane (Lemma 9). Our constant-approximation algorithm (Section 4.1) follows easily from the observation that there always exists a *binary* orthogonal partition whose “cost” is a constant factor of the optimal solution (Lemma 11), and we can compute such a partition in polynomial-time using dynamic programming. To obtain our subexponential-time algorithm, we observe that an orthogonal partition of the plane forms a planar graph, and so we can use a divide-and-conquer approach by utilizing balanced separators of this graph. Finally, our QPTAS (Section 4.2) is more complicated. It is also based on a planar separator theorem, together with a cutting lemma (Lemma 13) of [3]. The basic idea is to guess a balanced separator of the planar graph of the cutting and do divide-and-conquer. We then carefully analyze the quality of the solution computed by this divide-and-conquer process (Lemma 15 and Corollary 16), and show it is indeed a $(1 + \varepsilon)$ -approximation.

We begin with introducing some notations and concepts. Let $\Delta > 0$ be a sufficiently small number such that $3\Delta \leq |x_i - x_j|$ for all $i, j \in [n]$ with $x_i \neq x_j$ and $3\Delta \leq |x'_i - x'_j|$ for all $i, j \in [n]$ with $x'_i \neq x'_j$. Define $X = \{x_i - \Delta : i \in [n]\} \cup \{x_i + \Delta : i \in [n]\}$ and $X' = \{x'_i - \Delta : i \in [n]\} \cup \{x'_i + \Delta : i \in [n]\}$. We say a rectangle $[x_-, x_+] \times [x'_-, x'_+]$ is *regular* if $x_-, x_+ \in X \cup \{-\infty, \infty\}$ and $x'_-, x'_+ \in X' \cup \{-\infty, \infty\}$. Let \mathcal{R}_{reg} denote the set of all regular rectangles. The total number of different regular rectangles is $O(n^4)$, i.e., $|\mathcal{R}_{\text{reg}}| = O(n^4)$, because $|X| = O(n)$ and $|X'| = O(n)$. Note that if R is a regular rectangle, then for any $i \in [n]$, the point (x_i, x'_i) is either contained in the interior of R or outside R . We say a regular rectangle R is *nonempty* if $(x_i, x'_i) \in R$ for some $i \in [n]$, and *empty* otherwise. For a nonempty rectangle R , we define

$$\delta_R = 1 + \min_{f \in \mathbb{R}[x, x']_g} \sum_{(x_i, x'_i) \in R} (y_i - f(x_i, x'_i))^2.$$

Note that δ_R can be computed in $n^{O(1)}$ time using the standard approach for least-square polynomial regression. For a set \mathcal{R} of regular rectangles, denote by $\mathcal{R}_\bullet \subseteq \mathcal{R}$ the subset of nonempty rectangles, and define $\sigma_S(\mathcal{R}) = \sum_{R \in \mathcal{R}_\bullet} \delta_R$. A *regular region* refers to a subset of \mathbb{R}^2 that is the union of regular rectangles.

An *orthogonal partition* (OP) Π of a region $K \subseteq \mathbb{R}^2$ is a set of interior-disjoint (axis-parallel) rectangles whose union is K (see Figure 1 for an illustration). An OP Π is *regular* if all rectangles in Π are regular. The following lemma shows that our problem can be reduced to computing a regular OP Π of the plane which minimizes $\sigma_S(\Pi)$.



■ **Figure 1** An orthogonal partition (OP) of the region K .

► **Lemma 9.** *For any $f \in \Gamma_g^2$, there exists a regular OP Π of \mathbb{R}^2 such that $|\Pi| \leq 5|f| + 1$ and $\sigma_S(\Pi) \leq \sigma_S(f)$. Conversely, given a regular OP Π of \mathbb{R}^2 , one can compute in $n^{O(1)}$ time a function $f \in \Gamma_g^2$ such that $\sigma_S(f) = \sigma_S(\Pi)$.*

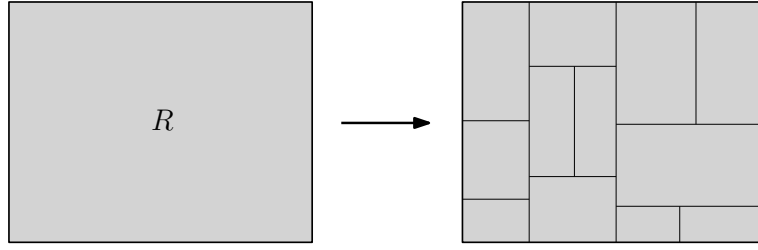
Using the reduction of Lemma 9, we establish our algorithms for piecewise polynomial regression for bivariate data. Section 4.1 presents a polynomial-time constant-approximation algorithm (Theorem 3), and Section 4.2 presents a QPTAS (Theorem 4).

4.1 A polynomial-time constant-approximation algorithm

In this section, we present a polynomial-time constant-approximation algorithm for the problem. Let Π^* be a regular OP of \mathbb{R}^2 that minimizes $\sigma_S(\Pi^*)$. In order to describe our algorithm, we need to introduce the notion of *binary* OP (and regular binary OP).

► **Definition 10** (binary OP). *Let R be an axis-parallel rectangle. A **binary OP** of R is an OP defined using the following recursive rule:*

- *The trivial partition $\{R\}$ is a binary OP of R .*



■ **Figure 2** A binary OP of the rectangle R .

- If ℓ is a horizontal or vertical line that partitions R into two smaller rectangles R_1 and R_2 , and Π_1 (resp., Π_2) are binary OPs of R_1 (resp., R_2), then $\Pi_1 \cup \Pi_2$ is a binary OP of R .

A binary OP is **regular** if it only consists of regular rectangles.

See Figure 2 for an illustration of binary OP. The basic idea of our approximation algorithm is to, instead of computing an optimal regular OP, compute an optimal *binary* regular OP, i.e., a regular binary OP Π of \mathbb{R}^2 that minimizes $\sigma_S(\Pi)$. This task can be solved in polynomial time by a simple dynamic programming algorithm as follows. Suppose we want to compute an optimal binary regular OP Π of a regular rectangle R . Then Π is either the trivial partition $\{R\}$ of R , or there exists a horizontal or vertical line ℓ separating R into two rectangles R_1 and R_2 , and $\Pi = \Pi_1 \cup \Pi_2$ where Π_1 (resp., Π_2) is a regular binary OPs of R_1 (resp., R_2). In the latter case, the equation of the line ℓ must be $x = \tilde{x}$ for some $\tilde{x} \in X$ or $x' = \tilde{x}'$ for some $\tilde{x}' \in X'$, because Π has to be a regular OP. This implies that R_1 and R_2 are regular rectangles. Furthermore, Π_1 and Π_2 must be optimal regular binary OPs of R_1 and R_2 , respectively, in order to minimize $\sigma_S(\Pi)$. Therefore, if we already know the optimal regular binary OPs of all regular rectangles R' such that $\text{area}(R') < \text{area}(R)$, then an optimal regular binary OPs of R can be computed in $O(n)$ time. The details of our algorithm is shown in Algorithm 2, which computes an optimal regular binary OP of \mathbb{R}^2 . Since $|\mathcal{R}_{\text{reg}}| = O(n^4)$, it is clear that Algorithm 2 runs in polynomial time.

Let Π_{bin} be the optimal regular binary OP of \mathbb{R}^2 computed by Algorithm 2 and Π^* be the regular OP of \mathbb{R}^2 that minimizes $\sigma_S(\Pi^*)$. We shall show that $\sigma_S(\Pi_{\text{bin}}) = O(\sigma_S(\Pi^*))$. To this end, we need the following two lemmas.

► **Lemma 11.** *For any regular OP Π of \mathbb{R}^2 , there exists a regular binary OP Π' of \mathbb{R}^2 such that $|\Pi'| = O(|\Pi_\bullet|)$ and for any $R' \in \Pi'_\bullet$ there exists $R \in \Pi_\bullet$ such that $R' \subseteq R$.*

► **Lemma 12.** *Let Π and Π' be two regular OP of \mathbb{R}^2 . If for any $R' \in \Pi'_\bullet$ there exists $R \in \Pi_\bullet$ such that $R' \subseteq R$, then we have $\sigma_S(\Pi') - \sigma_S(\Pi) \leq |\Pi'_\bullet| - |\Pi_\bullet|$.*

By Lemma 11, there exists a regular binary OP Π' of \mathbb{R}^2 such that $|\Pi'_\bullet| \leq O(|\Pi_\bullet^*|)$ and for any $R' \in \Pi'_\bullet$ there exists $R \in \Pi_\bullet^*$ such that $R' \subseteq R$. Then by Lemma 12, we have $\sigma_S(\Pi')/\sigma_S(\Pi^*) = 1 + (\sigma_S(\Pi') - \sigma_S(\Pi^*))/\sigma_S(\Pi^*) \leq 1 + (|\Pi'_\bullet| - |\Pi_\bullet^*|)/|\Pi_\bullet^*| = |\Pi'_\bullet|/|\Pi_\bullet^*| = O(1)$. Because Π_{bin} is an optimal regular binary OP of \mathbb{R}^2 , we further have $\sigma_S(\Pi_{\text{bin}}) \leq \sigma_S(\Pi') \leq O(\sigma_S(\Pi^*))$. We have $\sigma_S(\Pi^*) \leq \text{opt}$ by the first statement of Lemma 9, and hence $\sigma_S(\Pi_{\text{bin}}) \leq O(\text{opt})$. Using the second statement of Lemma 9, we then compute a function $f \in \Gamma_g^2$ in $O(n \cdot |\Pi_{\text{bin}}|) = O(n^5)$ time such that $\sigma_S(f) = \sigma_S(\Pi_{\text{bin}}) \leq O(\text{opt})$.

► **Theorem 3.** *There exists a constant-factor approximation algorithm for bivariate piecewise polynomial regression which runs in polynomial time.*

Algorithm 2 OPTBINPARTITION(S).

```

1:  $N \leftarrow |\mathcal{R}_{\text{reg}}|$ 
2: sort the rectangles in  $\mathcal{R}_{\text{reg}}$  as  $R_1, \dots, R_N$  such that  $\text{area}(R_1) \leq \dots \leq \text{area}(R_N)$ 
3: for  $i$  from 1 to  $N$  do
4:    $\Pi[R_i] \leftarrow \{R_i\}$  and  $\text{opt}[R_i] \leftarrow \sigma_S(\Pi[R_i])$ 
5:   suppose  $R_i = [x_-, x_+] \times [x'_-, x'_+]$ 
6:   for all  $z \in X$  such that  $x_- < z < x_+$  do
7:      $R'_i \leftarrow [x_-, z] \times [x'_-, x'_+]$  and  $R''_i \leftarrow [z, x_+] \times [x'_-, x'_+]$ 
8:     if  $\text{opt}[R_i] > \text{opt}[R'_i] + \text{opt}[R''_i]$  then
9:        $\Pi[R_i] \leftarrow \Pi[R'_i] \cup \Pi[R''_i]$  and  $\text{opt}[R_i] \leftarrow \sigma_S(\Pi[R_i])$ 
10:  for all  $z' \in X'$  such that  $x'_- < z' < x'_+$  do
11:     $R'_i \leftarrow [x_-, x_+] \times [x'_-, z']$  and  $R''_i \leftarrow [x_-, x_+] \times [z', x'_+]$ 
12:    if  $\text{opt}[R_i] > \text{opt}[R'_i] + \text{opt}[R''_i]$  then
13:       $\Pi[R_i] \leftarrow \Pi[R'_i] \cup \Pi[R''_i]$  and  $\text{opt}[R_i] \leftarrow \sigma_S(\Pi[R_i])$ 
14: return  $\Pi[\mathbb{R}^2]$ 

```

4.2 A quasi-polynomial-time approximation scheme

In this section, we design a quasi-polynomial-time approximation scheme (QPTAS) for the problem, that is, a $(1 + \varepsilon)$ -approximation algorithm which runs in $n^{\log^{O(1)} n}$ time for any fixed $\varepsilon > 0$. To this end, we borrow an idea from the geometric independent set literature [3, 4, 5, 13], which combines the cutting lemma and the planar separator theorem. We need the following cutting lemma.

► **Lemma 13.** *Given a set \mathcal{R} of interior-disjoint regular rectangles and a number $1 \leq r \leq |\mathcal{R}|$, there exists a regular OP Π of \mathbb{R}^2 with $|\Pi| = O(r)$ such that each rectangle in Π intersects at most $|\mathcal{R}|/r$ rectangles in \mathcal{R} .*

Proof. This lemma follows directly from a result of [3] (Lemma 3.12). The original statement in Lemma 3.12 of [3] only claims the existence of a partition Π of \mathbb{R}^2 satisfying the desired properties. However, by the construction in [3], if \mathcal{R} consists of regular rectangles, then the partition Π is a regular OP. ◀

Using the above cutting lemma and the (weighted) planar separator theorem, we obtain the following corollary.

► **Corollary 14.** *Given a set \mathcal{R} of interior-disjoint regular rectangles in \mathbb{R}^2 and a number $1 \leq r \leq |\mathcal{R}|$, there exists a set Σ of $O(\sqrt{r})$ interior-disjoint regular rectangles such that each rectangle in Σ intersects at most $|\mathcal{R}|/r$ rectangles in \mathcal{R} and for any regular region $K \subseteq \mathbb{R}^2$, the closure of each connected component U of $K \setminus (\bigcup_{R \in \Sigma} R)$ entirely contains at most $\frac{2}{3}|\mathcal{R}|$ rectangles in \mathcal{R} .*

Proof. We shall use the following weighted version of the planar separator theorem. Let $G = (V, E)$ be a planar graph with m vertices where each vertex has a non-negative weight, and W be the total weight of the vertices. The weighted planar separator theorem states that one can partition the vertex set V into three parts V_1, V_2, Σ such that (i) there is no edge between V_1 and V_2 , (ii) $|\Sigma| \leq O(\sqrt{m})$, and (iii) the total weight of the vertices in V_i is at most $\frac{2}{3}W$ for $i \in \{1, 2\}$.

Let Π be the regular partition of \mathbb{R}^2 described in Lemma 13 satisfying that $|\Pi| = O(r)$ and each rectangle in Π intersects at most $|\mathcal{R}|/r$ rectangles in \mathcal{R} . Consider the planar graph G_Π induced by Π . We assign each vertex of G_Π (i.e., each rectangle in Π) a non-negative

weight as follows. For each rectangle $R \in \mathcal{R}$, let $m(R)$ be the number of rectangles in Π that intersects R . The weight of each rectangle $R' \in \Pi$ is the sum of $1/m(R)$ for all $R \in \mathcal{R}$ that intersects R' . Note that the total weight W is equal to $|\mathcal{R}|$ because each rectangle in \mathcal{R} contributes exactly 1 to the total weight. Applying the weighted planar separator theorem to the vertex-weighted graph G_Π , we now partition Π into three parts V_1, V_2, Σ such that (i) there is no edge between V_1 and V_2 in G_Π , (ii) $|\Sigma| \leq O(\sqrt{r})$, and (iii) the total weight of the vertices in V_i is at most $\frac{2}{3}|\mathcal{R}|$ for $i \in \{1, 2\}$. The separator Σ is just the desired set of interior-disjoint regular rectangles described in the corollary. The fact that each rectangle in Σ intersects at most $|\mathcal{R}|/r$ rectangles in \mathcal{R} follows directly from the property of Π . So it suffices to show that for any regular region $K \subseteq \mathbb{R}^2$, (the closure of) each connected component of $K \setminus (\bigcup_{R \in \Sigma} R)$ intersects at most $\frac{2}{3}|\mathcal{R}|$ rectangles in \mathcal{R} . Let U be a connected component of $K \setminus (\bigcup_{R \in \Sigma} R)$. The rectangles in Π that are contained in the closure of U induces a connected subgraph of G_Π , and hence they either all belong to V_1 or all belong to V_2 (because there is no edge between V_1 and V_2 in G_Π). It follows that the total weight of these rectangles is at most $\frac{2}{3}|\mathcal{R}|$, which further implies that the number of rectangles in \mathcal{R} that are (entirely) contained in the closure of U is at most $\frac{2}{3}|\mathcal{R}|$. ◀

With the above corollary in hand, we are ready to describe our QPTAS. Roughly speaking, our algorithm “guesses” the set Σ in Corollary 14 for the optimal regular OP \mathcal{R} (and some parameter r polynomial in $\log n$ and $1/\varepsilon$) and then recursively solve the sub-problem in each rectangle in Σ and in each connected component of the complement of $\bigcup_{R \in \Sigma} R$. The nice properties of Σ described in Corollary 14 can be used to show (with a careful analysis) that the final solution we compute is a $(1 + \varepsilon)$ -approximation of the optimal solution.

Formally, let $r = \omega(1)$ be an integer parameter to be determined later and c be a sufficiently large constant. For a regular region $K \subseteq \mathbb{R}^2$ and an integer m , we denote by $\text{opt}_{K,m}$ as the minimum $\sigma_S(\Pi)$ for a regular OP Π of K with $|\Pi_\bullet| \leq m$. We shall design a procedure $\text{APPXPARTITION}(S, K, m)$, which computes a regular OP Π of the regular region K such that $\sigma_S(\Pi)$ is “not much larger” than $\text{opt}_{K,m}$ (note that we do *not* require $|\Pi_\bullet| \leq m$); what we mean by “not much larger” will be clear shortly.

■ **Algorithm 3** $\text{APPXPARTITION}(S, K, m)$.

```

1:  $\Pi_{\text{opt}} \leftarrow \emptyset$  and  $\text{opt} \leftarrow \infty$ 
2: for all  $\Pi \subseteq \mathcal{R}_{\text{reg}}$  with  $|\Pi| \leq r$  do
3:   if the rectangles in  $\Pi$  are interior-disjoint and contained in  $K$  then
4:     construct an arbitrary regular OP  $\Pi'$  of  $K$  such that  $\Pi \subseteq \Pi'$ 
5:     if  $\sigma_S(\Pi') < \text{opt}$  then  $\Pi_{\text{opt}} \leftarrow \Pi'$  and  $\text{opt} \leftarrow \sigma_S(\Pi')$ 
6: if  $m \leq r$  then return  $\Pi_{\text{opt}}$ 
7: for all  $\Sigma \subseteq \mathcal{R}_{\text{reg}}$  with  $|\Sigma| \leq c\sqrt{r}$  do
8:   if the rectangles in  $\Sigma$  are interior-disjoint then
9:      $\mathcal{U} \leftarrow \text{Components}(K \setminus (\bigcup_{R \in \Sigma} R))$ 
10:     $\Pi_R \leftarrow \text{APPXPARTITION}(S, K \cap R, m/r)$  for all  $R \in \Sigma$ 
11:     $\Pi_U \leftarrow \text{APPXPARTITION}(S, \text{Closure}(U), \frac{3}{4}m)$  for all  $U \in \mathcal{U}$ 
12:     $\Pi \leftarrow (\bigcup_{R \in \Sigma} \Pi_R) \cup (\bigcup_{U \in \mathcal{U}} \Pi_U)$ 
13:    if  $\sigma_S(\Pi) < \text{opt}$  then  $\Pi_{\text{opt}} \leftarrow \Pi$  and  $\text{opt} \leftarrow \sigma_S(\Pi)$ 
14: return  $\Pi_{\text{opt}}$ 

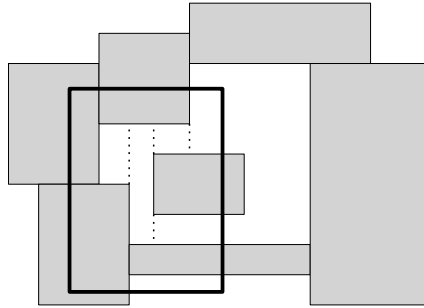
```

Algorithm 3 shows how $\text{APPXPARTITION}(S, K, m)$ works step-by-step, and here we provide an intuitive explanation of the algorithm. Let Π^* be a (unknown) regular OP of K such that $|\Pi^*| \leq m$ and $\sigma_S(\Pi^*) = \text{opt}_{K,m}$. We consider two cases separately: $|\Pi_\bullet^*| \leq r$ and

$|\Pi_\bullet^*| > r$. The for-loop of Line 2-6 handles the case $|\Pi_\bullet^*| \leq r$. We simply guess the (at most) r rectangles in Π_\bullet^* . Note that when we correctly guess Π_\bullet^* , i.e., $\Pi = \Pi_\bullet^*$ in Line 2, any regular OP Π' of K such that $\Pi \subseteq \Pi'$ satisfies $\sigma_S(\Pi') = \sigma_S(\Pi) = \sigma_S(\Pi_\bullet^*) = \sigma_S(\Pi^*)$, because $(x_i, x'_i) \notin K \setminus (\bigcup_{R \in \Pi} R)$ for all $i \in [n]$. Therefore, in the case $|\Pi_\bullet^*| \leq r$, we already have $|\Pi_{\text{opt}}| \leq \text{opt}_{K,m}$ after the for-loop of Line 2-6. The remaining case is $|\Pi_\bullet^*| > r$, which implies $m > r$. This case is handled in the for-loop of Line 8-15. We guess the set Σ described in Corollary 14 with $\mathcal{R} = \Pi_\bullet^*$ (Line 8 of Algorithm 3), which consists of at most $c\sqrt{r}$ interior-disjoint regular rectangles (recall that c is sufficiently large). Let \mathcal{U} be the set of connected components of $K \setminus (\bigcup_{R \in \Sigma} R)$. By Corollary 14, for each $R \in \Sigma$, the regular region $K \cap R$ intersects at most $|\Pi_\bullet^*|/r$ (and hence at most m/r) rectangles in \mathcal{R} , and for each $U \in \mathcal{U}$, the closure of U contains at most $\frac{2}{3}|\Pi_\bullet^*|$ rectangles (and hence at most $\frac{2}{3}m$) in \mathcal{R} . We then recursively call $\text{APPXPARTITION}(S, K \cap R, m/r)$ for all $R \in \Sigma$ and $\text{APPXPARTITION}(S, \text{Closure}(U), \frac{3}{4}m)$ for all $U \in \mathcal{U}$; see Line 11-12 of Algorithm 3. Each recursive call returns us a regular OP of the corresponding sub-region of K ; we set Π to be the union of all the returned regular OPs, which is clearly a regular OP of K (Line 13 of Algorithm 3). Intuitively, $\sigma_S(\Pi)$ should be “not much larger” than $\sigma_S(\Pi^*)$ if our guess for Σ is correct. More precisely, we have the following observation.

► **Lemma 15.** $\sum_{R \in \Sigma} \text{opt}_{K \cap R, m/r} + \sum_{U \in \mathcal{U}} \text{opt}_{\text{Closure}(U), \frac{3}{4}m} \leq (1 + O(1/\sqrt{r})) \cdot \sigma_S(\Pi^*)$.

Proof. We first show that there exists a regular OP Π of K satisfying (i) $|\Pi_\bullet| - |\Pi_\bullet^*| = O(|\Pi_\bullet^*|/\sqrt{r})$, (ii) each rectangle in Π is either contained in some $R \in \Sigma$ or interior-disjoint with all $R \in \Sigma$, (iii) each $R \in \Sigma$ contains at most m/r nonempty rectangles in Π and $\text{Closure}(U)$ contains at most $\frac{3}{4}m$ nonempty rectangles in Π for each $U \in \mathcal{U}$. Consider the regular OP Π^* of K . We further partition each rectangle $R^* \in \Pi^*$ into smaller (regular) rectangles as follows. Let $m(R^*)$ denote the number of rectangles in Σ that intersect (the interior of) R^* . Since the rectangles in Σ are interior-disjoint, the boundaries of these $m(R^*)$ rectangles cut R^* into $m(R^*) + 1$ regions (which are not necessarily rectangles). Now we construct the vertical decomposition the boundaries of these $m(R^*)$ rectangles inside R^* as follows (similarly to what we did in the proof of Lemma 9). For each top (resp., bottom) vertex of the $m(R^*)$ rectangles, if the vertex is contained in the interior of R^* , we shoot an upward (resp., downward) vertical ray from the vertex, which goes upwards (resp., downwards) until hitting the boundary of R^* or the boundary of some other $R \in \Sigma$. See Figure 3 for an illustration. Including one ray cuts R^* into one more piece, and the total number of the rays we shoot is at most $4m(R^*)$. Therefore, the vertical decomposition induces a regular OP of R^* into at most $5m(R^*) + 1$ rectangles. We do this for every rectangle $R^* \in \Pi^*$. After that, we obtain our desired regular OP Π . Next, we verify that Π satisfies



■ **Figure 3** The vertical decomposition inside R^* . The grey rectangles are those in Σ . The rectangle with bolder boundary is R^* .

the three conditions. We have $|\Pi_\bullet| \leq \sum_{R^* \in \Pi_\bullet^*} (5m(R^*) + 1) = \sum_{R^* \in \Pi_\bullet^*} 5m(R^*) + |\Pi_\bullet^*|$ since each rectangle $R^* \in \Pi_\bullet^*$ is partitioned into at most $5m(R^*) + 1$ smaller rectangles in Π (note that the rectangles in $\Pi^* \setminus \Pi_\bullet^*$ do not contribute any nonempty rectangle to Π). Because $|\Sigma| = O(\sqrt{r})$ and each rectangle in Σ intersects at most $|\Pi_\bullet^*|/r = |\Pi_\bullet^*|/r$ rectangles in Π_\bullet^* , we have $\sum_{R^* \in \Pi_\bullet^*} m(R^*) = O(|\Pi_\bullet^*|/\sqrt{r})$. It follows that $|\Pi_\bullet| - |\Pi_\bullet^*| = O(|\Pi_\bullet^*|/\sqrt{r})$, i.e., Π satisfies condition (i). Condition (ii) follows directly from our construction of Π . It suffices to show condition (iii). Let $R \in \Sigma$ be a rectangle. By our construction of Π , inside each $R^* \in \Pi^*$ that intersects (the interior of) R , there is exactly one rectangle in Π that is contained in R . Since R only intersects at most $|\Pi_\bullet^*|/r$ nonempty rectangles in Π^* and $|\Pi_\bullet^*| \leq m$, R contains at most m/r nonempty rectangles in Π . Let $U \in \mathcal{U}$ be a connected component of $K \setminus (\bigcup_{R \in \Sigma} R)$. Denote by $\Pi_\bullet^*(U) \subseteq \Pi_\bullet^*$ be the subset of rectangles that intersect U . Clearly, the number of nonempty rectangles in Π that are contained in $\text{Closure}(U)$ is at most $\sum_{R^* \in \Pi_\bullet^*(U)} (5m(R^*) + 1) = |\Pi_\bullet^*(U)| + O(|\Pi_\bullet^*|/\sqrt{r})$. By Corollary 14, $\text{Closure}(U)$ entirely contains at most $\frac{2}{3}|\Pi_\bullet^*(U)|$ rectangles in $\Pi_\bullet^*(U)$. All the other rectangles in $\Pi_\bullet^*(U)$ are partially contained in $\text{Closure}(U)$. Note that if a rectangle is partially contained in $\text{Closure}(U)$, then it intersects some $R \in \Sigma$. Therefore, the number of rectangles in $\Pi_\bullet^*(U)$ that are partially contained in $\text{Closure}(U)$ is bounded by $O(|\Pi_\bullet^*|/\sqrt{r})$, because $|\Sigma| = O(\sqrt{r})$ and each rectangle in Σ intersects at most $|\Pi_\bullet^*|/r$ rectangles in Π_\bullet^* . It follows that $|\Pi_\bullet^*(U)| = \frac{2}{3}|\Pi_\bullet^*(U)| + O(|\Pi_\bullet^*|/\sqrt{r})$ and the number of rectangles in Π that are contained in $\text{Closure}(U)$ is bounded by $\frac{2}{3}|\Pi_\bullet^*(U)| + O(|\Pi_\bullet^*|/\sqrt{r})$, which is no more than $\frac{3}{4}m$ because $|\Pi_\bullet^*| \leq m$ and we require $r = \omega(1)$.

Now we are ready to prove the lemma. Let Π be the regular OP of K we constructed above. Condition (ii) above guarantees that each rectangle in Π is either contained in some $R \in \Sigma$ or contained in $\text{Closure}(U)$ for some $U \in \mathcal{U}$. For each $R \in \Sigma$, let $\Pi(R) \subseteq \Pi$ denote the subset of rectangles contained in R . Similarly, for each $U \in \mathcal{U}$, let $\Pi(U) \subseteq \Pi$ denote the subset of rectangles contained in $\text{Closure}(U)$. Condition (iii) above guarantees that $|\Pi(R)_\bullet| \leq m/r$ for all $R \in \Sigma$ and $|\Pi(U)_\bullet| \leq \frac{3}{4}m$ for all $U \in \mathcal{U}$. So we have

$$\sigma_S(\Pi) = \sum_{R \in \Sigma} \sigma_S(\Pi(R)) + \sum_{R \in U \in \mathcal{U}} \sigma_S(\Pi(U)) \geq \sum_{R \in \Sigma} \text{opt}_{K \cap R, m/r} + \sum_{U \in \mathcal{U}} \text{opt}_{\text{Closure}(U), \frac{3}{4}m}.$$

On the other hand, we have $\sigma_S(\Pi) - \sigma_S(\Pi^*) \leq |\Pi_\bullet| - |\Pi_\bullet^*| = O(|\Pi_\bullet^*|/\sqrt{r})$ by Lemma 12 and condition (i) above. Because $|\Pi_\bullet^*| \leq \sigma_S(\Pi^*)$, we further have $\sigma_S(\Pi) \leq (1 + O(1/\sqrt{r})) \cdot \sigma_S(\Pi^*)$. Combining the two inequalities above gives us the inequality in the lemma. \blacktriangleleft

► **Corollary 16.** *Let Π_{opt} be the regular OP of K returned by $\text{APPXPARTITION}(S, K, m)$. Then we have $\sigma_S(\Pi_{\text{opt}}) \leq (1 + O(1/\sqrt{r}))^{O(\log m)} \cdot \text{opt}_{K, m}$.*

Proof. As before, let Π^* be a (unknown) regular OP of K such that $|\Pi_\bullet^*| \leq m$ and $\sigma_S(\Pi^*) = \text{opt}_{K, m}$. We prove that $\sigma_S(\Pi_{\text{opt}}) \leq (1 + O(1/\sqrt{r}))^{\log_{3/4} m} \cdot \text{opt}_{K, m}$ by induction on m . In the base case where $m \leq r$, we have $\sigma_S(\Pi_{\text{opt}}) \leq \sigma_S(\Pi^*) = \text{opt}_{K, m}$ after the for-loop of Line 2-6 (as argued before). Now suppose $m > r$. If $|\Pi_\bullet^*| \leq r$, then we still have $\sigma_S(\Pi_{\text{opt}}) \leq \text{opt}_{K, m}$ after the for-loop of Line 2-6 (as argued before). So it suffices to consider the case $|\Pi_\bullet^*| > r$. We show that when we correctly guess the set Σ in Line 8, the regular OP Π of K we construct in Line 13 satisfies $\sigma_S(\Pi) \leq (1 + O(1/\sqrt{r}))^{\log_{3/4} m} \cdot \text{opt}_{K, m}$. Let \mathcal{U} be the set of connected components of $K \setminus (\bigcup_{R \in \Sigma} R)$, as in Line 10. We have $\Pi = (\bigcup_{R \in \Sigma} \Pi_R) \cup (\bigcup_{U \in \mathcal{U}} \Pi_U)$ where $\Pi_R = \text{APPXPARTITION}(S, K \cap R, m/r)$ and $\Pi_U = \text{APPXPARTITION}(S, \text{Closure}(U), \frac{3}{4}m)$.

Recall that $r = \omega(1)$, and hence $m/r \leq \frac{3}{4}m$. By our induction hypothesis and Lemma 15,

$$\begin{aligned}
\sigma_S(\Pi) &= \sum_{R \in \Sigma} \sigma_S(\Pi_R) + \sum_{U \in \mathcal{U}} \sigma_S(\Pi_U) \\
&\leq (1 + O(1/\sqrt{r}))^{\log_{3/4} m - 1} \cdot \left(\sum_{R \in \Sigma} \text{opt}_{K \cap R, m/r} + \sum_{U \in \mathcal{U}} \text{opt}_{\text{Closure}(U), \frac{3}{4}m} \right) \\
&\leq (1 + O(1/\sqrt{r}))^{\log_{3/4} m - 1} \cdot (1 + O(1/\sqrt{r})) \cdot \sigma_S(\Pi^*) \\
&= (1 + O(1/\sqrt{r}))^{\log_{3/4} m} \cdot \sigma_S(\Pi^*),
\end{aligned}$$

which completes the proof. \blacktriangleleft

By Corollary 16, if we set $r = c' \cdot (\log^2 n / \varepsilon^2)$ for a sufficiently large constant c' , then for any regular region K and any $m = O(n)$, the procedure $\text{APPXPARTITION}(S, K, m)$ will return a regular partition Π_{opt} of K such that $\sigma_S(\Pi_{\text{opt}}) \leq (1 + \varepsilon) \cdot \text{opt}_{K, m}$. To solve our problem, we only need to call $\text{APPXPARTITION}(S, \mathbb{R}^2, 5n + 1)$, which will return a regular partition Π_{opt} of \mathbb{R}^2 such that $\sigma_S(\Pi_{\text{opt}}) \leq (1 + \varepsilon) \cdot \text{opt}_{\mathbb{R}^2, 5n+1}$. By the first statement of Lemma 9, we have $\text{opt}_{\mathbb{R}^2, 5n+1} \leq \text{opt}$. Therefore, it suffices to use the second statement of Lemma 9 to compute a function $f \in \Gamma_g^2$ such that $\sigma_S(f) = \sigma_S(\Pi_{\text{opt}}) \leq (1 + \varepsilon) \cdot \text{opt}$.

Time complexity. If $m \leq r$, the procedure $\text{APPXPARTITION}(S, K, m)$ takes $n^{O(r)} = n^{O(\log^2 n / \varepsilon^2)}$ time. In the case $m > r$, there are $n^{O(\sqrt{r})}$ sets Σ to be considered in Line 8. For each Σ , we have $c\sqrt{r}$ recursive calls in Line 11 and $n^{O(1)}$ recursive calls in Line 12, and all the other work in the for-loop of Line 8-15 can be done in $n^{O(1)}$ time. In addition, Line 1-6 takes $n^{O(r)}$ time. Therefore, if we use $T(m)$ to denote the running time of $\text{APPXPARTITION}(S, K, m)$, we have the recurrence

$$T(m) = \begin{cases} n^{O(\sqrt{r})} \cdot T(m/r) + n^{O(\sqrt{r})} \cdot T(\frac{3}{4}m) + n^{O(r)} & \text{if } m > r, \\ n^{O(r)} & \text{if } m \leq r, \end{cases}$$

which solves to $T(m) = n^{O(\sqrt{r} \log m + r)}$. Since our initial call is $\text{APPXPARTITION}(S, \mathbb{R}^2, 5n + 1)$, the total running time of our algorithm is $n^{O(\sqrt{r} \log n + r)} = n^{O(\log^2 n / \varepsilon^2)}$.

► **Theorem 4.** *There exists a QPTAS for bivariate piecewise polynomial regression.*

5 Conclusion and future work

In this paper, we studied the regression problem for univariate and bivariate data using piecewise polynomial functions. The loss of a k -piece polynomial function is measured as the sum of λk and its square error, where $\lambda \geq 0$ is a pre-specified parameter. For univariate data, we gave a $(1 + \varepsilon)$ -approximation algorithm that runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time, assuming the data points are pre-sorted. For bivariate data, we presented three results, a subexponential-time exact algorithm, a polynomial-time constant-approximation algorithm, and a QPTAS. Finally, for completeness, we also proved the problem for bivariate data is NP-hard.

Our work suggests several open problems and future research directions. The complexity of solving the problem exactly for the univariate data remains a challenging open problem. Is there a subquadratic time algorithm, or is there a (conditional or unconditional) near-quadratic lower bound? For bivariate data, does there exist a PTAS, namely, a polynomial-time $(1 + \varepsilon)$ -approximation algorithm for any fixed $\varepsilon > 0$? Finally, designing efficient approximation algorithms for regression problems with more than two variables is an interesting problem.

References

- 1 Jayadev Acharya, Ilias Diakonikolas, Chinmay Hegde, Jerry Zheng Li, and Ludwig Schmidt. Fast and near-optimal algorithms for approximating distributions by histograms. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 249–263, 2015.
- 2 Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. Fast algorithms for segmented regression. In *International Conference on Machine Learning*, pages 2878–2886, 2016.
- 3 Anna Adamaszek, Sarel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *Journal of the ACM (JACM)*, 66(4):1–40, 2019.
- 4 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *2013 IEEE 54th annual symposium on foundations of computer science*, pages 400–409. IEEE, 2013.
- 5 Anna Adamaszek and Andreas Wiese. A qptas for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 645–656. SIAM, 2014.
- 6 Pankaj K Agarwal, Sarel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.
- 7 Pankaj K Agarwal and Subhash Suri. Surface approximation and geometric partitions. *SIAM Journal on Computing*, 27(4):1016–1035, 1998.
- 8 Boris Aronov, Tetsuo Asano, Naoki Katoh, Kurt Mehlhorn, and Takeshi Tokuyama. Polyline fitting of planar points under min-sum criteria. *International journal of computational geometry & applications*, 16(02n03):97–116, 2006.
- 9 Ilias Diakonikolas, Jerry Li, and Anastasia Voloshinov. Efficient algorithms for multidimensional segmented regression. *arXiv preprint arXiv:2003.11086*, 2020.
- 10 Michael T Goodrich. Efficient piecewise-linear function approximation using the uniform metric: (preliminary version). In *Proceedings of the tenth annual Symposium on Computational geometry*, pages 322–331, 1994.
- 11 Sudipto Guha. On the space–time of optimal, approximate and streaming algorithms for synopsis construction problems. *The VLDB Journal*, 17(6):1509–1535, 2008.
- 12 S Louis Hakimi and Edward F Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, 53(2):132–136, 1991.
- 13 Sarel Har-Peled. Quasi-polynomial time approximation scheme for sparse subsets of polygons. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 120–129, 2014.
- 14 Hosagrahar Visvesvaraya Jagadish, Nick Koudas, S Muthukrishnan, Viswanath Poosala, Kenneth C Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB*, volume 98, pages 24–27, 1998.
- 15 Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.
- 16 Frederick Mosteller, John Wilder Tukey, et al. *Data analysis and regression: a second course in statistics*. Pearson, 1977.
- 17 DP Wang, NF Huang, HS Chao, and Richard CT Lee. Plane sweep algorithms for the polygonal approximation problems with applications. In *International Symposium on Algorithms and Computation*, pages 515–522. Springer, 1993.

Bidirectional String Anchors: A New String Sampling Mechanism

Grigorios Loukides ✉ 

Department of Informatics, King's College London, UK

Solon P. Pissis ✉ 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Abstract

The minimizers sampling mechanism is a popular mechanism for string sampling introduced independently by Schleimer et al. [SIGMOD 2003] and by Roberts et al. [Bioinf. 2004]. Given two positive integers w and k , it selects the lexicographically smallest length- k substring in every fragment of w consecutive length- k substrings (in every sliding window of length $w + k - 1$). Minimizers samples are approximately uniform, locally consistent, and computable in linear time. Although they do not have good worst-case guarantees on their size, they are often small in practice. They thus have been successfully employed in several string processing applications. Two main disadvantages of minimizers sampling mechanisms are: first, they also do not have good guarantees on the expected size of their samples for every combination of w and k ; and, second, indexes that are constructed over their samples do not have good worst-case guarantees for on-line pattern searches.

To alleviate these disadvantages, we introduce bidirectional string anchors (bd-anchors), a new string sampling mechanism. Given a positive integer ℓ , our mechanism selects the lexicographically smallest rotation in every length- ℓ fragment (in every sliding window of length ℓ). We show that bd-anchors samples are also approximately uniform, locally consistent, and computable in linear time. In addition, our experiments using several datasets demonstrate that the bd-anchors sample sizes decrease proportionally to ℓ ; and that these sizes are competitive to or smaller than the minimizers sample sizes using the analogous sampling parameters. We provide theoretical justification for these results by analyzing the expected size of bd-anchors samples.

We also show that by using any bd-anchors sample, we can construct, in near-linear time, an index which requires linear (extra) space in the size of the sample and answers on-line pattern searches in near-optimal time. We further show, using several datasets, that a simple implementation of our index is consistently faster for on-line pattern searches than an analogous implementation of a minimizers-based index [Grabowski and Raniszewski, *Softw. Pract. Exp.* 2017].

Finally, we highlight the applicability of bd-anchors by developing an efficient and effective heuristic for top- K similarity search under edit distance. We show, using synthetic datasets, that our heuristic is more accurate and more than one order of magnitude faster in top- K similarity searches than the state-of-the-art tool for the same purpose [Zhang and Zhang, KDD 2020].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, string sampling, text indexing, top- K similarity search

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.64

Supplementary Material *Software (Source Code)*: <https://github.com/solonas13/bd-anchors>

Funding *Grigorios Loukides*: This paper is part of the Leverhulme Trust RPG-2019-399 project. *Solon P. Pissis*: This paper is part of the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539. This paper is also part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229.

Acknowledgements We would like to thank Tomasz Kociumaka for pointing us to [42, Theorem 20] and Michelle Sweering for useful discussions.



© Grigorios Loukides and Solon P. Pissis;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 64; pp. 64:1–64:21



Leibniz International Proceedings in Informatics

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

1 Introduction

The notion of *minimizers*, introduced independently by Schleimer et al. [57] and by Roberts et al. [55], is a mechanism to sample a set of positions over an input string. The goal of this sampling mechanism is, given a string T of length n over an alphabet Σ of size σ , to simultaneously satisfy the following properties:

Property 1 (approximately uniform sampling): Every sufficiently long fragment of T has a representative position sampled by the mechanism.

Property 2 (local consistency): Exact matches between sufficiently long fragments of T are preserved unconditionally by having the same (relative) representative positions sampled by the mechanism.

In most practical scenarios, sampling the smallest number of positions is desirable, as long as Properties 1 and 2 are satisfied. This is because it leads to small data structures or fewer computations. Indeed, the minimizers sampling mechanism satisfies the property of approximately uniform sampling: given two positive integers w and k , it selects at least one length- k substring in every fragment of w consecutive length- k substrings (Property 1). Specifically, this is achieved by selecting the starting positions of the smallest length- k substrings in every $(w + k - 1)$ -long fragment, where smallest is defined by a choice of a total order on the universe of length- k strings. These positions are called the “minimizers”. Thus from similar fragments, similar length- k substrings are sampled (Property 2). In particular, if two strings have a fragment of length $w + k - 1$ in common, then they have at least one minimizer corresponding to the same length- k substring. Let us denote by $\mathcal{M}_{w,k}(T)$ the set of minimizers of string T . The following example illustrates the sampling.

► **Example 1.** The set $\mathcal{M}_{w,k}$ of minimizers for $w = k = 3$ for string $T = \text{aabaaabcbda}$ (using a 1-based index) is $\mathcal{M}_{3,3}(T) = \{1, 4, 5, 6, 7\}$ and for string $Q = \text{abaaa}$ is $\mathcal{M}_{3,3}(Q) = \{3\}$. Indeed Q occurs at position 2 in T ; and Q and $T[2..6]$ have the minimizers 3 and 4, respectively, which both correspond to string aaa of length $k = 3$.

The minimizers sampling mechanism is very versatile, and it has been employed in various ways in many different applications [45, 63, 19, 10, 30, 34, 35, 46, 36]. Since its inception, the minimizers sampling mechanism has undergone numerous theoretical and practical improvements [53, 10, 51, 50, 15, 21, 68, 36, 70] with a particular focus on minimizing the size of the residual sample; see Section 6 for a summary on this line of research. Although minimizers have been extensively and successfully used, especially in bioinformatics, we observe several inherent problems with setting the parameters w and k . In particular, although the notion of length- k substrings (known as *k-mers* or *k-grams*) is a widely-used string processing tool, we argue that, in the context of minimizers, it may be causing many more problems than it solves: it is not clear to us why one should use an extra sampling parameter k to effectively characterize a fragment of length $\ell = w + k - 1$ of T . In what follows, we describe some problems that may arise when setting the parameters w and k .

Indexing: The most widely-used approach is to index the selected minimizers using a hash table. The *key* is the selected length- k substring and the *value* is the list of positions it occurs. If one would like to use length- k' substrings for the minimizers with $\ell = w + k - 1 = w' + k' - 1$, for some $w' \neq w$ and $k' \neq k$, they should compute the new set $\mathcal{M}_{w',k'}(T)$ of minimizers and construct their new index based on $\mathcal{M}_{w',k'}$ from scratch.

Querying: To the best of our knowledge, no index based on minimizers can return in optimal or near-optimal time all occurrences of a pattern Q of length $|Q| \geq \ell = w + k - 1$ in T .

Sample Size: If one would like to minimize the number of selected minimizers, they should consider different total orders on the universe of length- k strings, which may complicate practical implementations, often scaling only up to a small k value, e.g. $k = 16$ [21]. On

the other hand, when k is fixed and w increases, the length- k substrings in a fragment become increasingly decoupled from each other, and that *regardless of the total order* we may choose. Unfortunately, this interplay phenomenon is inherent to minimizers. It is known that $k \geq \log_\sigma(w) + c$, for a fixed constant c , is a *necessary condition* for the existence of minimizers samples with expected size in $\mathcal{O}(n/w)$ [68]; see Section 6.

We propose the notion of bidirectional string anchors (bd-anchors) to alleviate these disadvantages. The bd-anchors is a mechanism that drops the sampling parameter k and its corresponding disadvantages. We only fix a parameter ℓ , which can be viewed as the length $w + k - 1$ of the fragments in the minimizers sampling mechanism. The *bd-anchor* of a string X of length ℓ is the lexicographically smallest rotation (cyclic shift) of X . We unambiguously characterize this rotation by its leftmost starting position in string XX . The set $\mathcal{A}_\ell(T)$ of the order- ℓ bd-anchors of string T is the set of bd-anchors of all length- ℓ fragments of T . It can be readily verified that bd-anchors satisfy Properties 1 and 2.

► **Example 2.** The set $\mathcal{A}_\ell(T)$ of bd-anchors for $\ell = 5$ for string $T = \text{aabaaabcbda}$ (using a 1-based index) is $\mathcal{A}_5(T) = \{4, 5, 6, 11\}$ and for string $Q = \text{abaaa}$, $\mathcal{A}_5(Q) = \{3\}$. Indeed Q occurs at position 2 in T ; and Q and $T[2..6]$ have the bd-anchors 3 and 4, respectively, which both correspond to the rotation **aaaab**.

Let us remark that *string synchronizing* sets, introduced by Kempa and Kociumaka [41], is another string sampling mechanism which may be employed to resolve the disadvantages of minimizers. Yet, it appears to be quite complicated to be efficient in practice. For instance, in [20], the authors used a simplified and specific definition of string synchronizing sets to design a space-efficient data structure for answering longest common extension queries.

We consider the word RAM model of computations with w -bit machine words, where $w = \Omega(\log n)$, for stating our results. We also assume throughout that string T is over alphabet $\Sigma = \{1, 2, \dots, n^{\mathcal{O}(1)}\}$, which captures virtually any real-world scenario. We measure space in terms of w -bit machine words. We make the following three specific contributions:

1. In Section 3 we state that the set $\mathcal{A}_\ell(T)$, for any $\ell > 0$ and any T of length n , can be constructed in $\mathcal{O}(n)$ time; and that the expected size of \mathcal{A}_ℓ for strings of length n , randomly generated by a memoryless source with identical letter probabilities, is in $\mathcal{O}(n/\ell)$, for any integer $\ell > 0$ (proofs are deferred to the full version of our work). The latter is in contrast to minimizers which achieve the expected bound of $\mathcal{O}(n/w)$ only when $k \geq \log_\sigma w + c$, for some constant c [68]. We then show, using five real datasets, that indeed the size of \mathcal{A}_ℓ decreases proportionally to ℓ ; that it is competitive to or smaller than $\mathcal{M}_{w,k}$, when $\ell = w + k - 1$; and that it is *much smaller* than $\mathcal{M}_{w,k}$ for *small* w values, which is practically important, as widely-used aligners that are based on minimizers will require less space and computation time if bd-anchors are used instead.
2. In Section 4 we show an index based on $\mathcal{A}_\ell(T)$, for any string T of length n and any integer $\ell > 0$, which answers on-line pattern searches in near-optimal time. In particular, for any constant $\epsilon > 0$, we show that our index supports the following trade-offs:
 - it occupies $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ extra space and reports all k occurrences of any pattern Q of length $|Q| \geq \ell$ given on-line in $\mathcal{O}(|Q| + (k+1) \log^\epsilon(|\mathcal{A}_\ell(T)|))$ time; or
 - it occupies $\mathcal{O}(|\mathcal{A}_\ell(T)| \log^\epsilon(|\mathcal{A}_\ell(T)|))$ extra space and reports all k occurrences of any pattern Q of length $|Q| \geq \ell$ given on-line in $\mathcal{O}(|Q| + \log \log(|\mathcal{A}_\ell(T)|) + k)$ time.

We also show that our index can be constructed in $\mathcal{O}(n + |\mathcal{A}_\ell(T)| \sqrt{\log(|\mathcal{A}_\ell(T)|)})$ time. We then show, using five real datasets, that a simple implementation of our index is *consistently faster* in on-line pattern searches than an analogous implementation of the minimizers-based index proposed by Grabowski and Raniszewski in [30].

3. In Section 5 we highlight the applicability of bd-anchors by developing an efficient and effective heuristic for top- K similarity search under edit distance. This is a fundamental and extensively studied problem [37, 9, 11, 44, 64, 67, 54, 61, 60, 17, 33, 65, 66] with applications in areas including bioinformatics, databases, data mining, and information retrieval. We show, using synthetic datasets, that our heuristic, which is based on the bd-anchors index, is *more accurate* and *more than one order of magnitude faster* in top- K similarity searches than the state-of-the-art tool proposed by Zhang and Zhang in [66].

In Section 2, we provide some preliminaries; and in Section 6 we discuss works related to minimizers. Let us stress that, although other works may be related to our contributions, we focus on comparing to minimizers because they are extensively used in applications.

2 Preliminaries

We start with some basic definitions and notation following [13]. An *alphabet* Σ is a finite nonempty set of elements called *letters*. A *string* $X = X[1] \dots X[n]$ is a sequence of *length* $|X| = n$ of letters from Σ . The *empty* string, denoted by ε , is the string of length 0. The fragment $X[i..j]$ of X is an *occurrence* of the underlying *substring* $S = X[i] \dots X[j]$. We also say that S occurs at *position* i in X . A *prefix* of X is a fragment of X of the form $X[1..j]$ and a *suffix* of X is a fragment of X of the form $X[i..n]$. The set of all strings over Σ (including ε) is denoted by Σ^* . The set of all length- k strings over Σ is denoted by Σ^k . Given two strings X and Y , the *edit distance* $d_E(X, Y)$ is the minimum number of edit operations (letter insertion, deletion, or substitution) transforming one string into the other.

Let M be a finite nonempty set of strings over Σ of total length m . We define the *trie* of M , denoted by $\text{TR}(M)$, as a deterministic finite automaton that recognizes M . Its set of states (nodes) is the set of prefixes of the elements of M ; the initial state (root node) is ε ; the set of terminal states (leaf nodes) is M ; and edges are of the form $(u, \alpha, u\alpha)$, where u and $u\alpha$ are nodes and $\alpha \in \Sigma$. The size of $\text{TR}(M)$ is thus $\mathcal{O}(m)$. The *compacted trie* of M , denoted by $\text{CT}(M)$, contains the root node, the branching nodes, and the leaf nodes of $\text{TR}(M)$. The term compacted refers to the fact that $\text{CT}(M)$ reduces the number of nodes by replacing each maximal branchless path segment with a single edge, and that it uses a fragment of a string $s \in M$ to represent the label of this edge in $\mathcal{O}(1)$ machine words. The size of $\text{CT}(M)$ is thus $\mathcal{O}(|M|)$. When M is the set of suffixes of a string Y , then $\text{CT}(M)$ is called the *suffix tree* of Y , and we denote it by $\text{ST}(Y)$. The suffix tree of a string of length n over an alphabet $\Sigma = \{1, \dots, n^{\mathcal{O}(1)}\}$ can be constructed in $\mathcal{O}(n)$ time [22].

Let us fix throughout a string $T = T[1..n]$ of length $|T| = n$ over an ordered alphabet Σ . Recall that we make the standard assumption of an integer alphabet $\Sigma = \{1, 2, \dots, n^{\mathcal{O}(1)}\}$.

We start by defining the notion of minimizers of T from [55] (the definition in [57] is slightly different). Given an integer $k > 0$, an integer $w > 0$, and the i th length- $(w + k - 1)$ fragment $F = T[i..i + w + k - 2]$ of T , we define the (w, k) -*minimizers* of F as the positions $j \in [i, i + w)$ where a lexicographically minimal length- k substring of F occurs. The set $\mathcal{M}_{w,k}(T)$ of (w, k) -minimizers of T is defined as the set of (w, k) -minimizers of $T[i..i + w + k - 2]$, for all $i \in [1, n - w - k + 2]$. The *density* of $\mathcal{M}_{w,k}(T)$ is defined as the quantity $|\mathcal{M}_{w,k}(T)|/n$. The following bounds are obtained trivially. The density of any minimizer scheme is at least $1/w$, since at least one (w, k) -minimizer is selected in each fragment, and at most 1, when every (w, k) -minimizer is selected.

If we waive the lexicographic order assumption, the set $\mathcal{M}_{w,k}(T)$ can be computed on-line in $\mathcal{O}(n)$ time, and if we further assume a constant-time computable function that gives us an *arbitrary* rank for each length- k substring in Σ^k in constant amortized time [36]. This can be

implemented, for instance, using a rolling hash function (e.g. Karp-Rabin fingerprints [39]), and the rank (total order) is defined by this function. We also provide here, for completeness, a simple off-line $\mathcal{O}(n)$ -time algorithm that uses a lexicographic order.

► **Theorem 3.** *The set $\mathcal{M}_{w,k}(T)$, for any integers $w, k > 0$ and any string T of length n , can be constructed in $\mathcal{O}(n)$ time.*

Proof. The underlying algorithm has two main steps. In the first step, we construct $\text{ST}(T)$, the suffix tree of T in $\mathcal{O}(n)$ time [22]. Using a depth-first search traversal of $\text{ST}(T)$ we assign at every position of T in $[1, n - k + 1]$ the lexicographic rank of $T[i..i + k - 1]$ among all the length- k strings occurring in T . This process clearly takes $\mathcal{O}(n)$ time as $\text{ST}(T)$ is an ordered structure; it yields an array R of size $n - k + 1$ with lexicographic ranks. In the second step, we apply a folklore algorithm, which computes the minimum elements in a sliding window of size w (cf. [36]) over R . The set of reported indices is $\mathcal{M}_{w,k}(T)$. ◀

3 Bidirectional String Anchors

We introduce the notion of bidirectional string anchors (bd-anchors). Given a string W , a string R is a *rotation* (or cyclic shift or conjugate) of W if and only if there exists a decomposition $W = UV$ such that $R = VU$, for a string U and a nonempty string V . We often characterize R by its starting position $|U| + 1$ in $WW = UVUV$. We use the term rotation interchangeably to refer to string R or to its identifier $(|U| + 1)$.

► **Definition 4** (Bidirectional anchor). *Given a string X of length $\ell > 0$, the bidirectional anchor (bd-anchor) of X is the lexicographically minimal rotation $j \in [1, \ell]$ of X with minimal j . The set of order- ℓ bd-anchors of a string T of length $n > \ell$, for some integer $\ell > 0$, is defined as the set $\mathcal{A}_\ell(T)$ of bd-anchors of $T[i..i + \ell - 1]$, for all $i \in [1, n - \ell + 1]$.*

The *density* of $\mathcal{A}_\ell(T)$ is defined as the quantity $|\mathcal{A}_\ell(T)|/n$. It can be readily verified that the bd-anchors sampling mechanism satisfies Properties 1 (approximately uniform sampling) and 2 (local consistency).

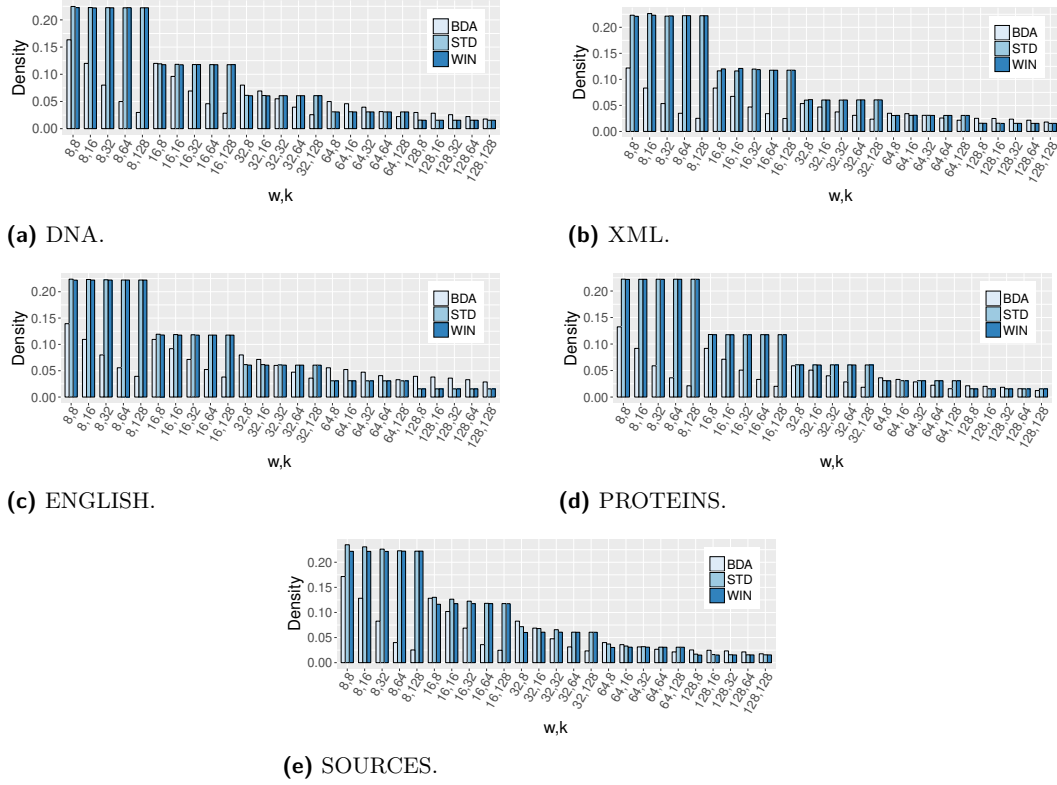
Construction and Size of \mathcal{A}_ℓ . We show that \mathcal{A}_ℓ admits an efficient construction. One can use the linear-time algorithm by Booth [6] to compute the lexicographically minimal rotation for each length- ℓ fragment of T , resulting in an $\mathcal{O}(n\ell)$ -time algorithm, which is reasonably fast for modest ℓ . (Booth's algorithm gives the leftmost minimal rotation by construction.) We can instead design an optimal $\mathcal{O}(n)$ -time algorithm for the construction of \mathcal{A}_ℓ , which is mostly of theoretical interest, via employing some elementary combinatorial observations and the data structure proposed by Kociumaka in [42, Theorem 20].

► **Theorem 5.** *The set $\mathcal{A}_\ell(T)$, for any integer $\ell > 0$ and any string T of length n , can be constructed in $\mathcal{O}(n)$ time.*

We can also show that the expected size of $\mathcal{A}_\ell(T)$ is in $\mathcal{O}(n/\ell)$ via observing that $\mathcal{A}_\ell(T)$ is expected to have many common elements with $\mathcal{M}_{w,k}(T)$, for certain values of w and k .

► **Theorem 6.** *If T is a string of length n , randomly generated by a memoryless source with identical letter probabilities, then, for any integer $\ell > 0$, the expected size of $\mathcal{A}_\ell(T)$ is in $\mathcal{O}(n/\ell)$.*

We defer the proofs of Theorems 5 and 6 to the full version of our work.



■ **Figure 1** Density vs. w, k for $\ell = w + k - 1$ and the datasets of Table 1.

Density Evaluation. We compare the density of bd-anchors, denoted by BDA, to the density of minimizers, for different values of w and k such that $\ell = w + k - 1$. This is a fair comparison because $\ell = w + k - 1$ is the length of the fragments considered by both mechanisms. We implemented bd-anchors, the standard minimizers mechanism from [55], and the minimizers mechanism with robust winnowing from [57], which are referred to as STD and WIN, respectively.

For bd-anchors, we used Booth’s algorithm, which is easy to implement and reasonably fast. For minimizers, we used Karp-Rabin fingerprints [39]. (Note that such “random” minimizers tend to perform *even better* than the ones based on lexicographic total order in terms of density [68].) Throughout, we do not evaluate construction times, as all implementations are reasonably fast, and we make the standard assumption that preprocessing is only required once. We used five string datasets from the popular Pizza & Chili corpus [24] (see Table 1 for the datasets characteristics). All implementations referred to in this paper have been written in C++ and compiled at optimization level -O3. All experiments reported in this paper were conducted using a single core of an AMD Opteron 6386 SE 2.8GHz CPU and 252GB RAM running GNU/Linux.

As can be seen by the results depicted in Figure 1, the density of bd-anchors is either significantly smaller than or competitive to the STD and WIN minimizers density, especially for small w . This is useful because a lower density results in smaller indexes and less computation (see Section 4), and because small w is of practical interest (see Section 5). For instance, the widely-used long-read aligner *Minimap2* [46] stores the selected minimizers of a reference genome in a hash table to find exact matches as anchors for seed-and-extend

■ **Table 1** Datasets characteristics.

Dataset	Length n	Alphabet size $ \Sigma $
DNA	200,000,000	4
XML	200,000,000	95
ENGLISH	200,000,000	224
PROTEINS	200,000,000	27
SOURCES	200,000,000	229

alignment. The parameters w and k are set based on the required sensitivity of the alignment, and thus w and k cannot be too large for high sensitivity. Thus, a lower sampling density reduces the size of the hash table, as well as the computation time, by lowering the average number of selected minimizers to consider when performing an alignment.

There exists a long line of research on improving the density of minimizers in special regimes (see Section 6 for details). We stress that most of these algorithms are designed, implemented, or optimized, *only for the DNA alphabet*. We have tested against two state-of-the-art tools employing such algorithms: Miniception [68] and PASHA [21]. The former did not give better results than STD or WIN for the tested values of w and k ; and the latter does not scale beyond $k = 16$ or with large alphabets. We have thus omitted these results.

We next report the average number (AVG) of bd-anchors of order $\ell \in \{4, 8, 12, 16\}$ over all strings of length $n = 20$ (see Table 2a) and over all strings of length $n = 32$ (see Table 2b), both over a binary alphabet. The results suggest that 2 may be a valid constant in $\mathcal{O}(n/\ell)$. As expected, the analogous AVG values using a ternary alphabet (not reported) were always lower than the corresponding ones with a binary alphabet.

■ **Table 2** Average number of bd-anchors for varying ℓ and: (a) $n = 20$ and (b) $n = 32$.

(a)					(b)				
(n, ℓ)	(20, 4)	(20, 8)	(20, 12)	(20, 16)	(n, ℓ)	(32, 4)	(32, 8)	(32, 12)	(32, 16)
$2n/\ell$	10	5	3.33	2.5	$2n/\ell$	16	8	5.33	4
AVG	8.53	4.37	2.77	1.76	AVG	14.16	7.67	5.26	3.85

4 Indexing Using Bidirectional Anchors

Before presenting our index, let us start with a basic definition that is central to our querying process.

► **Definition 7** ((α, β) -hit). *Given an order- ℓ bd-anchor $j_Q \in \mathcal{A}_\ell(Q)$, for some integer $\ell > 0$, of a query string Q , two integers $\alpha > 0, \beta > 0$, with $\alpha + \beta \geq \ell + 1$, and an order- ℓ bd-anchor $j_T \in \mathcal{A}_\ell(T)$ of a target string T , the ordered pair (j_Q, j_T) is called an (α, β) -hit if and only if $T[j_T - \alpha + 1 .. j_T] = Q[j_Q - \alpha + 1 .. j_Q]$ and $T[j_T .. j_T + \beta - 1] = Q[j_Q .. j_Q + \beta - 1]$.*

Intuitively, the parameters α and β let us choose a fragment of Q that is anchored at j_Q .

We would like to construct a data structure over T , which is based on $\mathcal{A}_\ell(T)$, such that, when we are given an order- ℓ bd-anchor j_Q over Q as an on-line query, together with parameters α and β , we can report all (α, β) -hits efficiently. To this end, we present an efficient data structure, denoted by $\mathcal{I}_\ell(T)$, which is constructed on top of T , and answers (α, β) -hit queries in near-optimal time. We prove the following result.

► **Theorem 8.** *Given a string T of length n and an integer $\ell > 0$, the $\mathcal{I}_\ell(T)$ index can be constructed in $\mathcal{O}(n + |\mathcal{A}_\ell(T)|\sqrt{\log(|\mathcal{A}_\ell(T)|)})$ time. For any constant $\epsilon > 0$, $\mathcal{I}_\ell(T)$:*

- *occupies $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ extra space and reports all k (α, β) -hits in $\mathcal{O}(\alpha + \beta + (k + 1)\log^\epsilon(|\mathcal{A}_\ell(T)|))$ time; or*

- occupies $\mathcal{O}(|\mathcal{A}_\ell(T)| \log^\epsilon(|\mathcal{A}_\ell(T)|))$ extra space and reports all k (α, β) -hits in $\mathcal{O}(\alpha + \beta + \log \log(|\mathcal{A}_\ell(T)|) + k)$ time.

Let us denote by $\overleftarrow{X} = X[|X|] \dots X[1]$ the *reversal* of string X . We now describe our data structure.

Construction of $\mathcal{I}_\ell(T)$. Given $\mathcal{A}_\ell(T)$, we construct two sets $\mathcal{S}_\ell^L(T)$ and $\mathcal{S}_\ell^R(T)$ of strings; conceptually, the reversed suffixes going *left* from j to 1, and the suffixes going *right* from j to n , for all j in $\mathcal{A}_\ell(T)$. In particular, for the bd-anchor j , we construct two strings: $\overleftarrow{T[1..j]} \in \mathcal{S}_\ell^L(T)$ and $T[j..n] \in \mathcal{S}_\ell^R(T)$. Note that, $|\mathcal{S}_\ell^L(T)| = |\mathcal{S}_\ell^R(T)| = |\mathcal{A}_\ell(T)|$, since for every bd-anchor in $\mathcal{A}_\ell(T)$ we have a distinct string in $\mathcal{S}_\ell^L(T)$ and in $\mathcal{S}_\ell^R(T)$.

We construct two *compacted tries* $\mathcal{T}_\ell^L(T)$ and $\mathcal{T}_\ell^R(T)$ over $\mathcal{S}_\ell^L(T)$ and $\mathcal{S}_\ell^R(T)$, respectively, to index all strings. Every string is concatenated with some special letter $\$$ not occurring in T , which is lexicographically minimal, to make $\mathcal{S}_\ell^L(T)$ and $\mathcal{S}_\ell^R(T)$ prefix-free (this is standard for conceptual convenience). The leaf nodes of the compacted tries are *labeled* with the corresponding j : there is a one-to-one correspondence between a leaf node and a bd-anchor j . In $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ time, we also enhance the nodes of the tries with a perfect static dictionary [26] to ensure constant-time retrieval of edges by the first letter of their label. Let $\mathcal{L}_\ell^L(T)$ denote the list of the leaf labels of $\mathcal{T}_\ell^L(T)$ as they are visited using a depth-first search traversal. $\mathcal{L}_\ell^L(T)$ corresponds to the (labels of the) lexicographically sorted list of $\mathcal{S}_\ell^L(T)$ in increasing order. For each node u in $\mathcal{T}_\ell^L(T)$, we also store the corresponding interval $[x_u, y_u]$ over $\mathcal{L}_\ell^L(T)$. Analogously for R , $\mathcal{L}_\ell^R(T)$ denotes the list of the leaf labels of $\mathcal{T}_\ell^R(T)$ as they are visited using a depth-first search traversal and corresponds to the (labels of the) lexicographically sorted list of $\mathcal{S}_\ell^R(T)$ in increasing order. For each node v in $\mathcal{T}_\ell^R(T)$, we also store the corresponding interval $[x_v, y_v]$ over $\mathcal{L}_\ell^R(T)$.

The total size occupied by the tries is $\Theta(|\mathcal{A}_\ell(T)|)$ because they are compacted: we label the edges with intervals over $[1, n]$ from T .

We also construct a 2D range reporting data structure over the following points in set $\mathcal{R}_\ell(T)$:

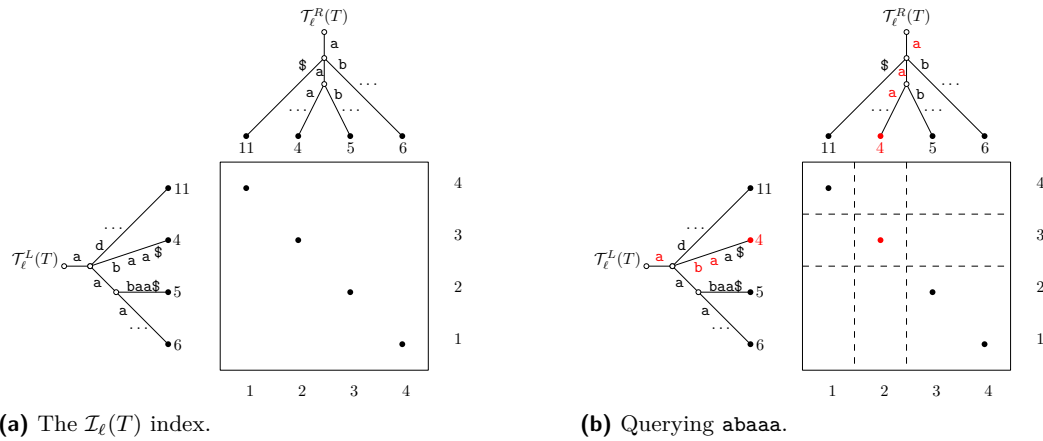
$$(x, y) \in \mathcal{R}_\ell(T) \iff \mathcal{L}_\ell^L(T)[x] = \mathcal{L}_\ell^R(T)[y].$$

Note that $|\mathcal{R}_\ell(T)| = |\mathcal{A}_\ell(T)|$ because the set of leaf labels stored in both tries is precisely the set $\mathcal{A}_\ell(T)$. Let us remark that the idea of employing 2D range reporting for bidirectional pattern searches has been introduced by Amir et al. [2] for text indexing and dictionary matching with one error; see also [47].

This completes the construction of $\mathcal{I}_\ell(T)$. We next explain how we can query $\mathcal{I}_\ell(T)$.

Querying. Given a bd-anchor j_Q over a string Q as an on-line query and parameters $\alpha, \beta > 0$, we spell $\overleftarrow{Q[j_Q - \alpha + 1..j_Q]}$ in $\mathcal{T}_\ell^L(T)$ and $Q[j_Q..j_Q + \beta - 1]$ in $\mathcal{T}_\ell^R(T)$ starting from the root nodes. If any of the two strings is not spelled fully, we return no (α, β) -hits. If both strings are fully spelled, we arrive at node u in $\mathcal{T}_\ell^L(T)$ (resp. v in $\mathcal{T}_\ell^R(T)$), which corresponds to an interval over $\mathcal{L}_\ell^L(T)$ stored in u (resp. $\mathcal{L}_\ell^R(T)$ in v). We obtain the two intervals $[x_u, y_u]$ and $[x_v, y_v]$ forming a rectangle and ask the corresponding 2D range reporting query. It can be readily verified that this query returns all (α, β) -hits.

► **Example 9.** Let $T = \text{aabaaabcbda}$ and $\mathcal{A}_5(T) = \{4, 5, 6, 11\}$. We have the following strings in $\mathcal{S}^L(T)$: $\overleftarrow{T[1..4]} = \text{abaa}$; $\overleftarrow{T[1..5]} = \text{aabaa}$; $\overleftarrow{T[1..6]} = \text{aaabaa}$; and $\overleftarrow{T[1..11]} = \text{adbcbaaabaa}$. We have the following strings in $\mathcal{S}^R(T)$: $T[4..11] = \text{aaabcbda}$; $T[5..11] = \text{aabcbda}$; $T[6..11] = \text{abcbda}$; $T[11..11] = \text{a}$. Inspect Figure 2.



■ **Figure 2** Let $T = \text{aabaaabcbda}$ and $\ell = 5$. Further let $Q = \text{aacabaaaae}$, the bd-anchor $6 \in \mathcal{A}_5(Q)$ of order 5 corresponding to $Q[4..8]$, $\alpha = 3$ and $\beta = 3$. The figure illustrates the $\mathcal{I}_\ell(T)$ index and how we find that $Q[4..8] = T[2..5] = \text{abaaa}$: the fragment $T[2..5]$ is anchored at position 4.

Proof of Theorem 8. We use the $\mathcal{O}(n)$ -time algorithm underlying Theorem 5 to construct $\mathcal{A}_\ell(T)$. We use the $\mathcal{O}(n)$ -time algorithm from [3, 8] to construct the compacted tries from $\mathcal{A}_\ell(T)$. We extract the $|\mathcal{A}_\ell(T)|$ points $(x, y) \in \mathcal{R}_\ell(T)$ using the compacted tries in $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ time. For the first trade-off of the statement, we use the $\mathcal{O}(|\mathcal{A}_\ell(T)|\sqrt{\log(|\mathcal{A}_\ell(T)|)})$ -time algorithm from [5] to construct the 2D range reporting data structure over $\mathcal{R}_\ell(T)$ from [7]. For the second trade-off, we use the $\mathcal{O}(|\mathcal{A}_\ell(T)|\sqrt{\log(|\mathcal{A}_\ell(T)|)})$ -time algorithm from [28] to construct the 2D range reporting data structure over $\mathcal{R}_\ell(T)$ from the same paper. ◀

We obtain the following corollary for the fundamental problem of *text indexing* [62, 48, 22, 38, 23, 31, 32, 4, 12, 52, 41, 27].

► **Corollary 10.** *Given $\mathcal{I}_\ell(T)$ constructed for some integer $\ell > 0$ and some constant $\epsilon > 0$ over string T , we can report all k occurrences of any pattern Q , $|Q| \geq \ell$, in T in time:*

- $\mathcal{O}(|Q| + (k + 1) \log^\epsilon(|\mathcal{A}_\ell(T)|))$ when $\mathcal{I}_\ell(T)$ occupies $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ extra space; or
- $\mathcal{O}(|Q| + \log \log(|\mathcal{A}_\ell(T)|) + k)$ when $\mathcal{I}_\ell(T)$ occupies $\mathcal{O}(|\mathcal{A}_\ell(T)| \log^\epsilon(|\mathcal{A}_\ell(T)|))$ extra space.

Proof. Every occurrence of Q in T is prefixed by string $P = Q[1.. \ell]$. We first compute the bd-anchor of P in $\mathcal{O}(\ell)$ time using Booth's algorithm. Let this bd-anchor be j . We set $\alpha = j$ and $\beta = |Q| - j + 1$. The result follows by applying Theorem 8. ◀

Index Evaluation. Consider a hash table with the following (key, value) pairs: the *key* is the hash value $h(S)$ of a length- k string S ; and the *value* (satellite data) is a list of occurrences of S in T . It should be clear that such a hash table indexing the minimizers of T does not perform well for on-line pattern searches of *arbitrary length* because it would need to verify the remaining prefix and suffix of the pattern using letter comparisons *for all occurrences* of a minimizer in T . We thus opted for comparing our index to the one of [30], which addresses this specific problem by sampling the suffix array [48] with minimizers to reduce the number of letter comparisons during verification.

To ensure a fair comparison, we have implemented the basic index from [30]; we denote it by GR Index. We used Karp-Rabin [39] fingerprints for computing the minimizers of T . We also used the array-based version of the suffix tree that consists of the suffix array (SA) and the longest common prefix (LCP) array [48]; SA was constructed using SDSL [29] and the LCP array using the Kasai et al. [40] algorithm.

We sampled the SA using the minimizers. Given a pattern Q , we searched $Q[j \dots |Q|]$ starting with the minimizer $Q[j \dots j + k - 1]$ using the Manber and Myers [48] algorithm on the sampled SA. For verifying the remaining prefix $Q[1 \dots j - 1]$ of Q , we used letter comparisons, as described in [30]. The space complexity of this implementation is $\mathcal{O}(n)$ and the extra space for the index is $\mathcal{O}(|\mathcal{M}_{w,k}(T)|)$. The query time is not bounded. We have implemented two versions of our index. We used Booth's algorithm for computing the bd-anchors of T . We used SDSL for SA construction and the Kasai et al. algorithm for LCP array construction. We sampled the SA using the bd-anchors thus constructing $\mathcal{L}_\ell^L(T)$ and $\mathcal{L}_\ell^R(T)$. Then, the two versions of our index are:

1. **BDA Index v1:** Let j be the bd-anchor of $Q[1 \dots \ell]$. For $\overleftarrow{Q[1 \dots j]}$ (resp. $Q[j \dots |Q|]$) we used the Manber and Myers algorithm for searching over $\mathcal{L}_\ell^L(T)$ (resp. $\mathcal{L}_\ell^R(T)$). We used range trees [14] implemented in CGAL [59] for 2D range reporting as per the described querying process. The space complexity of this implementation is $\mathcal{O}(n + |\mathcal{A}_\ell(T)| \log(|\mathcal{A}_\ell(T)|))$ and the extra space for the index is $\mathcal{O}(|\mathcal{A}_\ell(T)| \log(|\mathcal{A}_\ell(T)|))$. The query time is $\mathcal{O}(|Q| + \log^2(|\mathcal{A}_\ell(T)|) + k)$, where k is the total number of occurrences of Q in T .
2. **BDA Index v2:** Let j be the bd-anchor of $Q[1 \dots \ell]$. If $|Q| - j + 1 \geq j$ (resp. $|Q| - j + 1 < j$), we search for $Q[j \dots |Q|]$ (resp. $\overleftarrow{Q[1 \dots j]}$) using the Manber and Myers algorithm on $\mathcal{L}_\ell^R(T)$ (resp. $\mathcal{L}_\ell^L(T)$). For verifying the remaining part of the pattern we used letter comparisons. The space complexity of this implementation is $\mathcal{O}(n)$ and the extra space for the index is $\mathcal{O}(|\mathcal{A}_\ell(T)|)$. The query time is not bounded.

For each of the five real datasets of Table 1 and each query string length ℓ , we randomly extracted 500,000 substrings from the text and treated each substring as a query, following [30]. We plot the average query time in Figure 3. As can be seen, BDA Index v2 consistently outperforms GR Index across all datasets and all ℓ values. The better performance of BDA Index v2 is due to two theoretical reasons. First, the verification strategy exploits the fact that the index is *bidirectional* to apply the Manber and Myers algorithm to the largest part of the pattern, which results in fewer letter comparisons. Second, bd-anchors generally have smaller density compared to minimizers; see Figure 4. We also plot the peak memory usage in Figure 5. As can be seen, BDA Index v2 requires a similar amount of memory to GR Index.

BDA Index v1 was slower than GR Index for small ℓ but faster for large ℓ in three out of five datasets used and had by far the highest memory usage. Let us stress that the inefficiency of BDA Index v1 is not due to inefficiency in the query time or space of our algorithm. It is merely because the range tree implementation of CGAL, which is a standard off-the-shelf library, is unfortunately inefficient in terms of both query time and memory usage; see also [58, 25]. As BDA Index v2 was very efficient in all aspects, we defer the investigation of alternative range tree implementations [58] for BDA Index v1 to the full version of our work.

Discussion. The proposed $\mathcal{I}_\ell(T)$ index, which is based on bd-anchors, has the following attributes:

1. **Construction:** $\mathcal{A}_\ell(T)$ is constructed in $\mathcal{O}(n)$ worst-case time and $\mathcal{I}_\ell(T)$ is constructed in $\mathcal{O}(n + |\mathcal{A}_\ell(T)| \sqrt{\log(|\mathcal{A}_\ell(T)|)})$ worst-case time. These time complexities are near-linear in n and do not depend on the alphabet Σ as long as $|\Sigma| = n^{\mathcal{O}(1)}$, which is true for virtually any real scenario.
2. **Index Size:** By Theorem 8, $\mathcal{I}_\ell(T)$ can occupy $\mathcal{O}(|\mathcal{A}_\ell(T)|)$ space. By Theorem 6, the size of $\mathcal{A}_\ell(T)$ is $\mathcal{O}(n/\ell)$ in expectation and so $\mathcal{I}_\ell(T)$ can also be of size $\mathcal{O}(n/\ell)$. In practice this depends on T and on the implementation of the 2D range reporting data structure.
3. **Querying:** The $\mathcal{I}_\ell(T)$ index answers on-line pattern searches in near-optimal time.

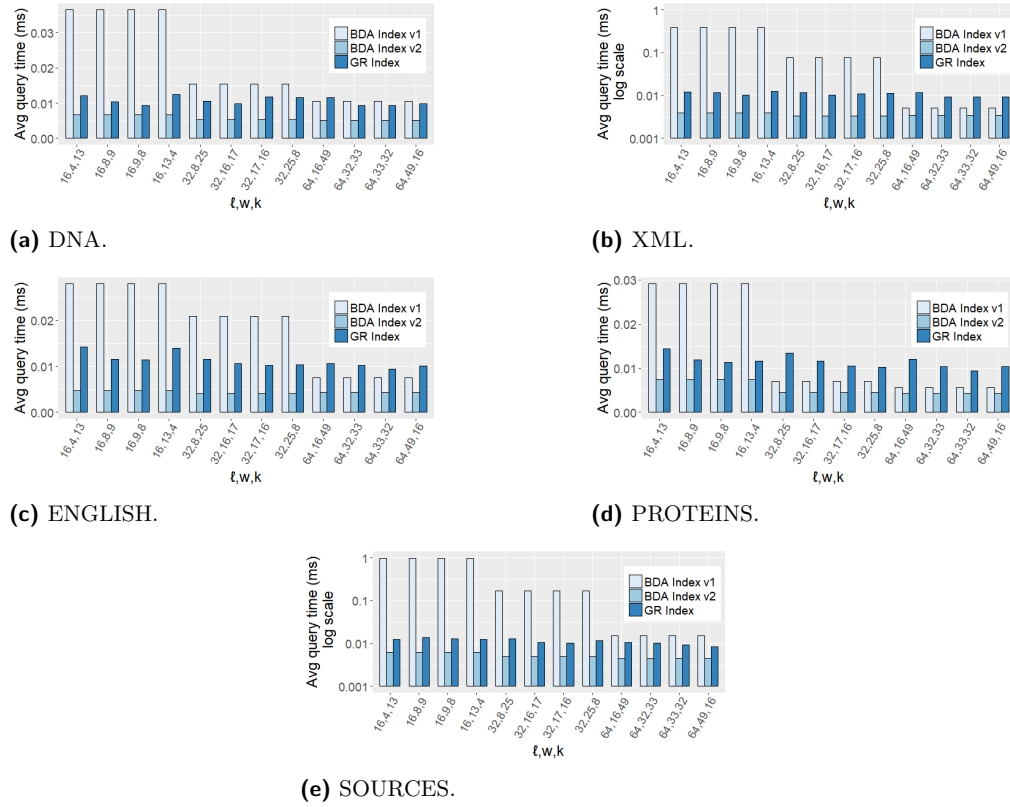


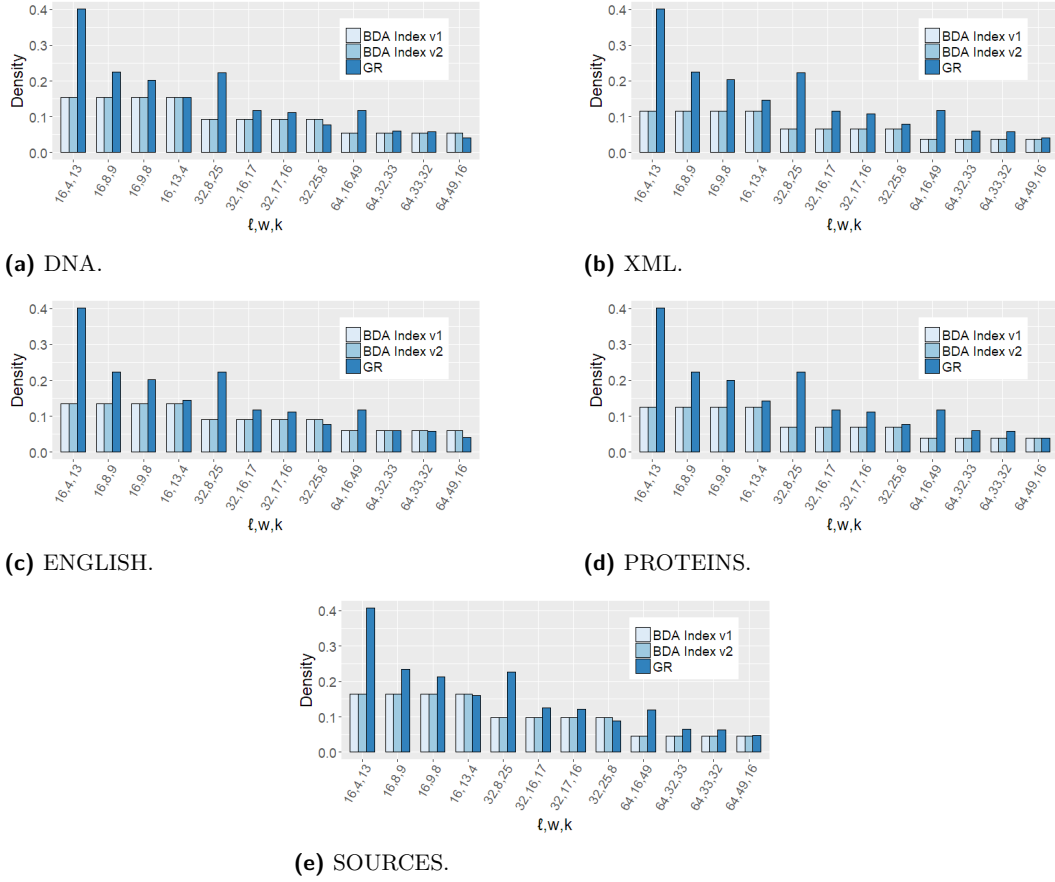
Figure 3 Average query time (ms) vs. w, k for $\ell = w + k - 1$ and the datasets of Table 1.

4. **Flexibility:** Note that one would have to *reconstruct* a (hash-based) index, which indexes the set of (w, k) -minimizers, to increase specificity or sensitivity: increasing k increases the specificity and decreases the sensitivity. Our $\mathcal{I}_\ell(T)$ index, conceptually truncated at string depth k , is essentially an index based on (w, k) -minimizers, which additionally wrap around. We can thus increase *specificity* by considering larger α, β values or increase *sensitivity* by considering smaller α, β values. This effect can be realized *without reconstructing* our $\mathcal{I}_\ell(T)$ index: we just adapt α and β upon querying accordingly.

5 Top- K Similarity Search under Edit Distance

We show how bd-anchors can be applied to speed up similarity search under edit distance. This is a fundamental problem with myriad applications in bioinformatics, databases, data mining, and information retrieval. It has thus been studied extensively in the literature both from a theoretical and a practical point of view [37, 9, 11, 44, 64, 67, 54, 61, 60, 17, 33, 65, 66]. Let \mathcal{D} be a collection of strings called *dictionary*. We focus, in particular, on indexing \mathcal{D} for answering the following type of top- K queries: Given a query string Q and an integer K , return K strings from the dictionary that are closest to Q with respect to edit distance. We follow a typical seed-chain-align approach as used by several bioinformatics applications [1, 16, 45, 46]. The main new ingredients we inject, with respect to this classic approach, is that we use: (1) bd-anchors as seeds; and (2) \mathcal{I}_ℓ to index the dictionary \mathcal{D} , for some integer parameter $\ell > 0$.

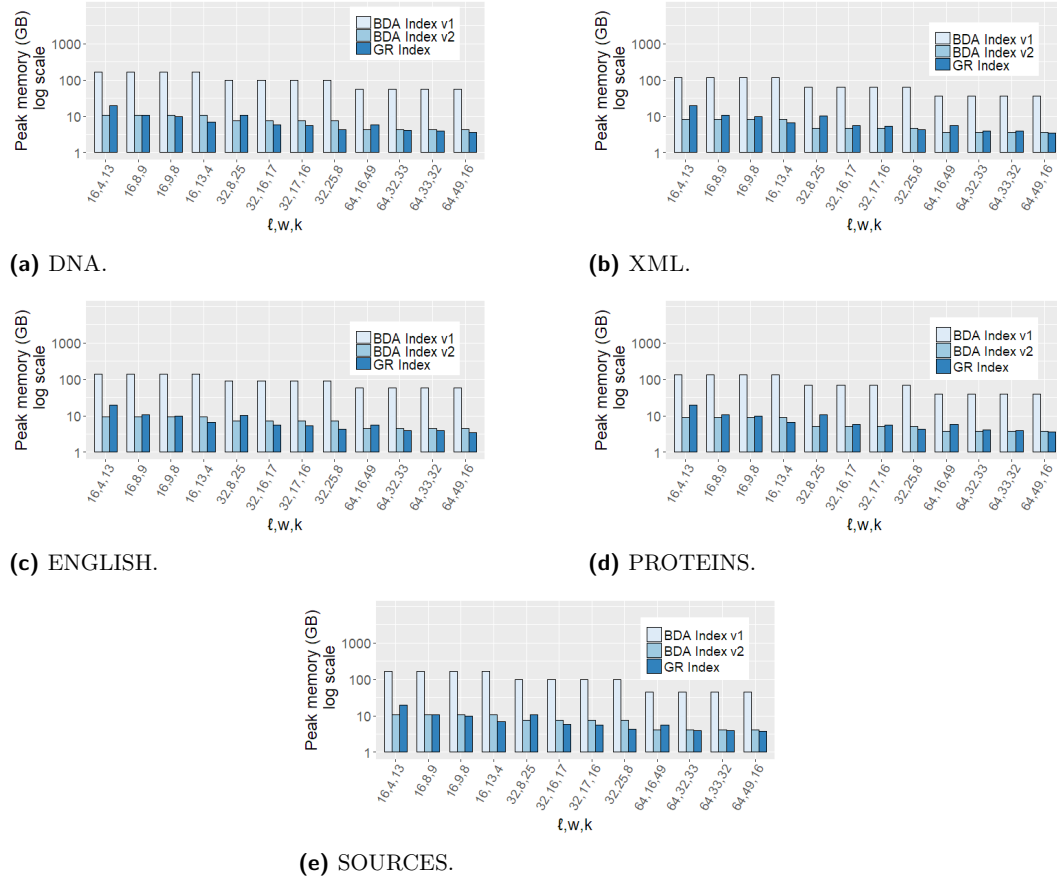
Construction. We require an integer parameter $\ell > 0$ defining the order of the bd-anchors. We set $T = S_1 \dots S_{|\mathcal{D}|}$, where $S_i \in \mathcal{D}$, compute the bd-anchors of order ℓ of T , and construct the $\mathcal{I}_\ell(T)$ index (see Section 4) using the bd-anchors.



■ **Figure 4** Density vs. w, k for $\ell = w + k - 1$ and the datasets of Table 1.

Querying. We require two parameters $\tau \geq 0$ and $\delta \geq 0$. The former parameter controls the sensitivity of our filtering step (Step 2 below); and the latter one controls the sensitivity of our verification step (Step 3 below). Both parameters trade accuracy for speed.

1. For each query string Q , we compute the bd-anchors of order ℓ . For every bd-anchor j_Q , we take an arbitrary fragment (e.g. the leftmost) of length ℓ anchored at j_Q as the *seed*. Let this fragment start at position i_Q . This implies a value for α and β , with $\alpha + \beta = \ell + 1$; specifically for $Q[i_Q \dots i_Q + \ell - 1]$ we have $Q[i_Q \dots j_Q] = Q[j_Q - \alpha + 1 \dots j_Q]$ and $Q[j_Q \dots i_Q + \ell - 1] = Q[j_Q \dots j_Q + \beta - 1]$. For every bd-anchor j_Q , we query $\overleftarrow{Q}[j_Q - \alpha + 1 \dots j_Q]$ in $\mathcal{T}_\ell^L(T)$ and $Q[j_Q \dots j_Q + \beta - 1]$ in $\mathcal{T}_\ell^R(T)$ and collect all (α, β) -hits.
2. Let $\tau \geq 0$ be an input parameter and let $L_{Q,S} = (q_1, s_1), \dots, (q_k, s_k)$ be the list of all (α, β) -hits between the queried fragments of string Q and fragments of a string $S \in \mathcal{D}$. If $h < \tau$, we consider string S as not found. The intuition here is that if Q and S are sufficiently close with respect to edit distance, they would have a relatively long $L_{Q,S}$ [16]. If $h \geq \tau$, we sort the elements of $L_{Q,S}$ with respect to their first component. (This comes for free because we process Q from left to right.) We then compute a *longest increasing subsequence* (LIS) in $L_{Q,S}$ with respect to the second component, which *chains* the (α, β) -hits, in $\mathcal{O}(h \log h)$ time [56] per $L_{Q,S}$ list. We use the LIS of $L_{Q,S}$ to *estimate* the *identity score* (total number of matching letters in a fixed alignment) for Q and S , which we denote by $E_{Q,S}$, based on the occurrences of the (α, β) -hits in the LIS.



■ **Figure 5** Peak memory usage (GB) vs. w, k for $\ell = w + k - 1$ and the datasets of Table 1.

- Let $\delta \geq 0$ be an input parameter and let E_K be the K th largest estimated identity score. We extract, as candidates, the ones whose estimated identity score is at least $E_K - \delta$. For every candidate string S , we close the gaps between the occurrences of the (α, β) -hits in the LIS using dynamic programming [43], thus computing an *upper bound* on the edit distance between Q and S (UB score). In particular, closing the gaps consists in summing up the exact edit distance for all pairs of fragments (one from S and one from Q) that lie in between the (α, β) -hits. We return K strings from the list of candidates with the lowest UB score. If $\delta = 0$, we return K strings with the highest $E_{Q,S}$ score.

Index Evaluation. We compared our algorithm, called **BDA Search**, to **Min Search**, the state-of-the-art tool for top- K similarity search under edit distance proposed by Zhang and Zhang in [66]. The main concept used in **Min Search** is the rank of a letter in a string, defined as the size of the neighborhood of the string in which the letter has the minimum hash value. Based on this concept, **Min Search** partitions each string in the dictionary \mathcal{D} into a hierarchy of substrings and then builds an index comprised of a set of hash tables, so that strings having common substrings and thus small edit distance are grouped into the same hash table. To find the top- K closest strings to a query string, **Min Search** partitions the query string based on the ranks of its letters and then traverses the hash tables comprising the index. Thanks to the index and the use of several filtering tricks, **Min Search** is at least one order of magnitude faster with respect to query time than popular alternatives [65, 67, 18].

We implemented two versions of BDA Search: BDA Search v1 which is based on BDA Index v1; and BDA Search v2 which is based on BDA Index v2. For Min Search, we used the C++ implementation from <https://github.com/kedayuge/Search>.

We constructed synthetic datasets, referred to as SYN, in a way that enables us to study the impact of different parameters and efficiently identify the ground truth (top- K closest strings to a query string with respect to edit distance). Specifically, we first generated 50 query strings and then constructed a cluster of K strings around each query string. To generate the query strings, we started from an arbitrary string Q of length $|Q| = 1000$ from a real dataset of protein sequences, used in [66], and generated a string Q' that is at edit distance e from Q , by performing e edit distance operations, each with equal probability. Then, we treated Q' as Q and repeated the process to generate the next query string. To create the clusters, we first added each query string into an initially empty cluster and then added $K - 1$ strings, each at edit distance at most $e' < e$ from the query string. The strings were generated by performing at most e' edit distance operations, each with equal probability. Thus, each cluster contains the top- K closest strings to the query string of the cluster. We used $K \in \{5, 10, 15, 20, 25\}$, $d = \frac{e}{|Q|} \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$, and $d' = \frac{e'}{|Q|} = d - 0.05$. We evaluated query answering accuracy using the F1 score [49], expressed as the harmonic mean of precision and recall¹. For BDA Search, we report results for $\tau = 0$ (full sensitivity during filtering) and $\delta = 0$ (no sensitivity during verification), as it was empirically determined to be a reasonable trade-off between accuracy and speed. For Min Search, we report results using its default parameters from [66].

We plot the F1 scores and average query time in Figures 6 and 7, respectively. All methods achieved almost perfect accuracy, in all tested cases. BDA Search slightly outperformed Min Search (by up to 1.1%), remaining accurate even for large ℓ ; the changes to F1 score for Min Search as ℓ varies are because the underlying method is randomized. However, both versions of BDA Search were *more than one order of magnitude faster* than Min Search on average (over all results of Figure 7), with BDA Search v1 being 2.9 times slower than BDA Search v2 on average, due to the inefficiency of the range tree implementation of CGAL. Furthermore, both versions of BDA Search scaled better with respect to K . For example, the average query time for BDA Search v1 became 2 times larger when K increased from 5 to 25 (on average over ℓ values), while that for Min Search became 5.4 times larger on average. The reason is that verification in Min Search, which increases the accuracy of this method, becomes increasingly expensive as K gets larger. The peak memory usage for these experiments is reported in Figure 8. Although Min Search outperforms BDA Search in terms of memory usage, BDA Search v2 still required a very small amount of memory (less than 1GB). BDA Search v1 required more memory for the reasons mentioned in Section 4.

Discussion. BDA Search outperforms Min Search in accuracy while being more than one order of magnitude faster in query time. These results are very encouraging because the efficiency of BDA Search is entirely due to injecting bd-anchors and not due to any further filtering tricks such as those employed by Min Search. Min Search clearly outperforms BDA Search in memory usage, albeit the memory usage of BDA Search v2 is still quite modest. We defer an experimental evaluation using real datasets to the full version of our work.

¹ Precision is the ratio between the number of returned strings that are among the top- K closest strings to a query string and the number of all returned strings. Recall is the ratio between the number of returned strings that are among the top- K closest strings to a query string and K . Since all tested algorithms return K strings, the F1 score in our experiments is equal to precision and equal to recall.

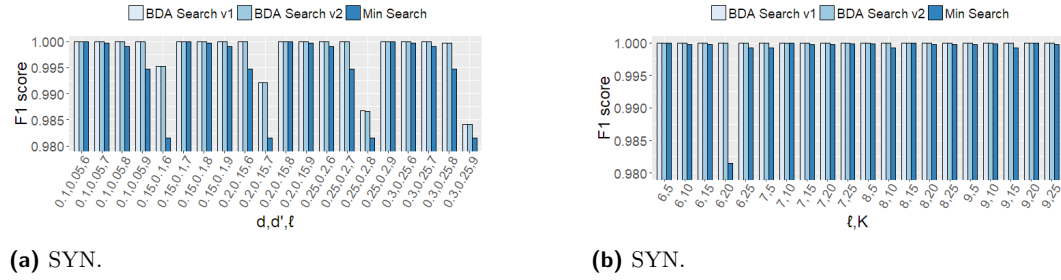


Figure 6 F1 score vs. (a) d, d', l , for $K = 20$, and (b) l, K , for $d = 0.15$ and $d' = 0.1$.

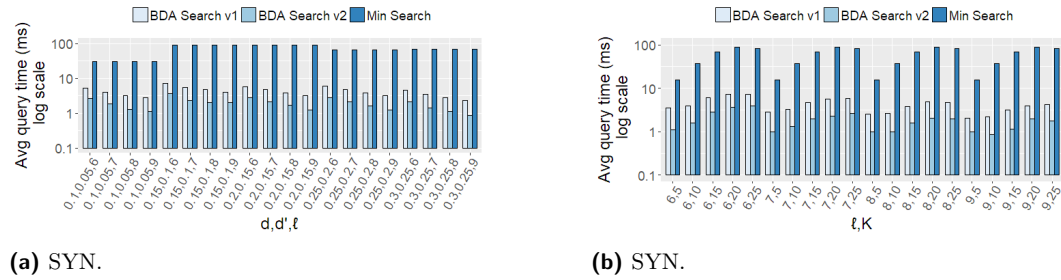
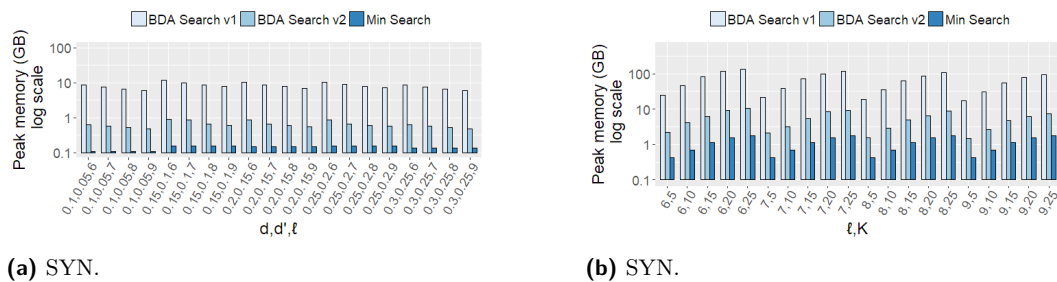


Figure 7 Average query time (ms) vs. (a) d, d', l , for $K = 20$, and (b) l, K , for $d = 0.15$ and $d' = 0.1$.

6 Other Works on Improving Minimizers

Although every sampling mechanism based on minimizers primarily aims at satisfying Properties 1 and 2, different mechanisms employ total orders that lead to substantially different total numbers of selected minimizers. Thus, research on minimizers has focused on determining total orders which lead to the lowest possible density (recall that the density is defined as the number of selected length- k substrings over the length of the input string). In fact, much of the literature focuses on the *average case* [53, 51, 50, 21, 68]; namely, the lowest expected density when the input string is random. In practice, many works use a “random minimizer” where the order is defined by choosing a permutation of all the length- k strings at random (e.g., by using a hash function, such as the Karp-Rabin fingerprints [39], on the length- k strings). Such a randomized mechanism has the benefit of being easy to implement and providing good expected performance in practice.

Minimizers and Universal Hitting Sets. A *universal hitting set* (UHS) is an unavoidable set of length- k strings, i.e., it is a set of length- k strings that “hits” every $(w + k - 1)$ -long fragment of every possible string. The theory of universal hitting sets [53, 50, 41, 69] plays an important role in the current theory for minimizers with low density on average. In particular, if a UHS has small size, it generates minimizers with a provable upper-bound on their density. However, UHSs are less useful in the string-specific case for two reasons [70]: (1) the requirement that a UHS has to hit every $(w + k - 1)$ -long fragment of every possible string is too strong; and (2) UHSs are too large to provide a meaningful upper-bound on the density in the string-specific case. Therefore, since in many practical scenarios the input string is known and does not change frequently, we try to optimize the density for one particular string instead of optimizing the average density over a random input.



■ **Figure 8** Peak memory usage (GB) vs. (a) d, d', ℓ , for $K = 20$, and (b) ℓ, K , for $d = 0.15$ and $d' = 0.1$.

String-Specific Minimizers. In the string-specific case, minimizers sampling mechanisms may employ frequency-based orders [10, 36]. In these orders, length- k strings occurring less frequently in the string compare less than the ones occurring more frequently. The intuition [70] is to obtain a sparse sampling by selecting infrequent length- k strings which should be spread apart in the string. However, there is no theoretical guarantee that a frequency-based order gives low density minimizers (there are many counter-examples). Furthermore, frequency-based orders do not always give minimizers with lower density in practice. For instance, the two-tier classification (very frequent vs. less frequent length- k strings) in the work of [36] outperforms an order that strictly follows frequency of occurrence.

A different approach to constructing string-specific minimizers is to start from a UHS and to remove elements from it, as long as it still hits every $(w + k - 1)$ -long fragment of the input string [15]. Since this approach starts with a UHS that is not related to the string, the improvement in density may not be significant [70]. Additionally, current methods [21] employing this approach are computationally limited to using $k \leq 16$, as the size of the UHS increases exponentially with k . Using such small k values may not be appropriate in some applications.

Other Improvements. When $k \approx w$, minimizers with expected density of $1.67/w + o(1/w)$ on a random string can be constructed using the approach of [68]. Such minimizers have guaranteed expected density less than $2/(w + 1)$ and work for infinitely many w and k . The approach of [68] also does not require the use of expensive heuristics to precompute and store a large set of length- k strings, unlike some methods [53, 15, 21] with low density in practice.

The notion of *polar set*, which can be seen as complementary to that of UHS, was recently introduced in [70]. While a UHS is a set of length- k strings that intersect with every $(w + k - 1)$ -long fragment at least once, a polar set is a set of length- k strings that intersect with any fragment at most once. The construction of a polar set builds upon sets of length- k strings that are sparse in the input string. Thus, the minimizers derived from these polar sets have provably tight bounds on their density. Unfortunately, computing optimal polar sets is NP-hard, as shown in [70]. Thus, the work of [70] also proposed a heuristic for computing feasible “good enough” polar sets. A main disadvantage of this approach is that when each length- k string occurs frequently in the input string, it becomes hard to select many length- k strings without violating the polar set condition.

References

- 1 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. doi: 10.1016/S0022-2836(05)80360-2.

- 2 Amihoud Amir, Dmitry Keselman, Gad M. Landau, Moshe Lewenstein, Noa Lewenstein, and Michael Rodeh. Text indexing and dictionary matching with one error. *J. Algorithms*, 37(2):309–325, 2000. doi:10.1006/jagm.2000.1104.
- 3 Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Inf. Comput.*, 270, 2020. doi:10.1016/j.ic.2019.104462.
- 4 Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 148–193, 2014. doi:10.1145/2591796.2591885.
- 5 Djamal Belazzougui and Simon J. Puglisi. Range predecessor and lempel-ziv parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.ch143.
- 6 Kellogg S. Booth. Lexicographically least circular substrings. *Inf. Process. Lett.*, 10(4/5):240–242, 1980. doi:10.1016/0020-0190(80)90149-0.
- 7 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.
- 8 Panagiotis Charalampopoulos, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. Property suffix array with applications in indexing weighted sequences. *ACM J. Exp. Algorithmics*, 25, 2020. doi:10.1145/3385898.
- 9 Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 313–324. ACM, 2003. doi:10.1145/872757.872796.
- 10 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinform.*, 32(12):201–208, 2016. doi:10.1093/bioinformatics/btw279.
- 11 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don’t cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004. doi:10.1145/1007352.1007374.
- 12 Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/s00453-013-9860-6.
- 13 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
- 14 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 15 Dan F. DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, and Guillaume Marçais. Practical universal k-mer sets for minimizer schemes. In Xinghua Mindy Shi, Michael Buck, Jian Ma, and Pierangelo Veltri, editors, *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB 2019, Niagara Falls, NY, USA, September 7-10, 2019*, pages 167–176. ACM, 2019. doi:10.1145/3307339.3342144.
- 16 Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, January 1999. doi:10.1093/nar/27.11.2369.
- 17 Dong Deng, Guoliang Li, and Jianhua Feng. A pivotal prefix based filtering algorithm for string similarity search. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International*

- Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 673–684. ACM, 2014. doi:10.1145/2588555.2593675.
- 18 Dong Deng, Guoliang Li, Jianhua Feng, and Wen-Syan Li. Top-k string similarity search with edit-distance constraints. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 925–936. IEEE Computer Society, 2013. doi:10.1109/ICDE.2013.6544886.
 - 19 Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. KMC 2: fast and resource-frugal k -mer counting. *Bioinform.*, 31(10):1569–1576, 2015. doi:10.1093/bioinformatics/btv022.
 - 20 Patrick Dinklage, Johannes Fischer, Alexander Herlez, Tomasz Kociumaka, and Florian Kurpicz. Practical Performance of Space Efficient Data Structures for Longest Common Extensions. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.39.
 - 21 Baris Ekim, Bonnie Berger, and Yaron Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In Russell Schwartz, editor, *Research in Computational Molecular Biology - 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10-13, 2020, Proceedings*, volume 12074 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2020. doi:10.1007/978-3-030-45257-5_3.
 - 22 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, 1997. doi:10.1109/SFCS.1997.646102.
 - 23 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
 - 24 Paolo Ferragina and Gonzalo Navarro. Pizza&Chili corpus – compressed indexes and their testbeds. <http://pizzachili.dcc.uchile.cl/texts.html>.
 - 25 Vissarion Fisikopoulos. An implementation of range trees with fractional cascading in C++. *CoRR*, abs/1103.4521, 2011. arXiv:1103.4521.
 - 26 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $\mathcal{O}(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
 - 27 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.
 - 28 Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.54.
 - 29 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, volume 8504 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 2014. doi:10.1007/978-3-319-07959-2_28.
 - 30 Szymon Grabowski and Marcin Raniszewski. Sampled suffix array with minimizers. *Softw. Pract. Exp.*, 47(11):1755–1771, 2017. doi:10.1002/spe.2481.
 - 31 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

- 32 Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Breaking a time-and-space barrier in constructing full-text indices. *SIAM J. Comput.*, 38(6):2162–2178, 2009. doi:10.1137/070685373.
- 33 Huiqi Hu, Guoliang Li, Zhifeng Bao, Jianhua Feng, Yongwei Wu, Zhiguo Gong, and Yaoqiang Xu. Top-k spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(2):551–565, 2016. doi:10.1109/TKDE.2015.2485213.
- 34 Chirag Jain, Alexander T. Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. *J. Comput. Biol.*, 25(7):766–779, 2018. doi:10.1089/cmb.2018.0036.
- 35 Chirag Jain, Sergey Koren, Alexander T. Dilthey, Adam M. Phillippy, and Srinivas Aluru. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinform.*, 34(17):i748–i756, 2018. doi:10.1093/bioinformatics/bty597.
- 36 Chirag Jain, Arang Rhie, Haowen Zhang, Claudia Chu, Brian Walenz, Sergey Koren, and Adam M. Phillippy. Weighted minimizer sampling improves long read mapping. *Bioinform.*, 36(Supplement-1):i111–i118, 2020. doi:10.1093/bioinformatics/btaa435.
- 37 Tamer Kahveci and Ambuj K. Singh. Efficient index structures for string databases. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 351–360. Morgan Kaufmann, 2001. URL: <http://www.vldb.org/conf/2001/P351.pdf>.
- 38 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.
- 39 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.
- 40 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsu Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In Amihood Amir and Gad M. Landau, editors, *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1-4, 2001 Proceedings*, volume 2089 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2001. doi:10.1007/3-540-48194-X_17.
- 41 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.
- 42 Tomasz Kociumaka. Minimal suffix and rotation of a substring in optimal time. 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 28:1–28:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.28.
- 43 Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.
- 44 Chen Li, Bin Wang, and Xiaochun Yang. VGRAM: improving performance of approximate queries on string collections using variable-length grams. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 303–314. ACM, 2007. URL: <http://www.vldb.org/conf/2007/papers/research/p303-li.pdf>.
- 45 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, March 2016. doi:10.1093/bioinformatics/btw152.
- 46 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinform.*, 34(18):3094–3100, 2018. doi:10.1093/bioinformatics/bty191.

- 47 Veli Mäkinen and Gonzalo Navarro. Position-restricted substring searching. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 703–714. Springer, 2006. doi:10.1007/11682462_64.
- 48 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.
- 49 Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.
- 50 Guillaume Marçais, Dan F. DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinform.*, 34(13):i13–i22, 2018. doi:10.1093/bioinformatics/bty258.
- 51 Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the performance of minimizers and winnowing schemes. *Bioinform.*, 33(14):i110–i117, 2017. doi:10.1093/bioinformatics/btx235.
- 52 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 408–424, 2017. doi:10.1137/1.9781611974782.26.
- 53 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Carl Kingsford. Compact universal k-mer hitting sets. In Martin C. Frith and Christian Nørgaard Storm Pedersen, editors, *Algorithms in Bioinformatics - 16th International Workshop, WABI 2016, Aarhus, Denmark, August 22-24, 2016. Proceedings*, volume 9838 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 2016. doi:10.1007/978-3-319-43681-4_21.
- 54 Jianbin Qin, Wei Wang, Chuan Xiao, Yifei Lu, Xuemin Lin, and Haixun Wang. Asymmetric signature schemes for efficient exact edit similarity query processing. *ACM Trans. Database Syst.*, 38(3):16:1–16:44, 2013. doi:10.1145/2508020.2508023.
- 55 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinform.*, 20(18):3363–3369, 2004. doi:10.1093/bioinformatics/bth408.
- 56 Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961. doi:10.4153/CJM-1961-015-3.
- 57 Saul Schleimer, Daniel Shawcross Wilkerson, and Alexander Aiken. Winnowing: Local algorithms for document fingerprinting. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 76–85. ACM, 2003. doi:10.1145/872757.872770.
- 58 Yihan Sun and Guy E. Blelloch. Parallel range, segment and rectangle queries with augmented maps. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 159–173. SIAM, 2019. doi:10.1137/1.9781611975499.13.
- 59 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2.1 edition, 2021. URL: <https://doc.cgal.org/5.2.1/Manual/packages.html>.
- 60 Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 85–96. ACM, 2012. doi:10.1145/2213836.2213847.
- 61 Xiaoli Wang, Xiaofeng Ding, Anthony K. H. Tung, and Zhenjie Zhang. Efficient and effective KNN sequence search with approximate n-grams. *Proc. VLDB Endow.*, 7(1):1–12, 2013. doi:10.14778/2732219.2732220.
- 62 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, 1973. doi:10.1109/SWAT.1973.13.

- 63 Derrick E. Wood and Steven L. Salzberg. Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3), 2014. Copyright: Copyright 2014 Elsevier B.V., All rights reserved. doi:10.1186/gb-2014-15-3-r46.
- 64 Zhenglu Yang, Jianjun Yu, and Masaru Kitsuregawa. Fast algorithms for top-k approximate string matching. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1939>.
- 65 Minghe Yu, Jin Wang, Guoliang Li, Yong Zhang, Dong Deng, and Jianhua Feng. A unified framework for string similarity search with edit-distance constraint. *VLDB J.*, 26(2):249–274, 2017. doi:10.1007/s00778-016-0449-y.
- 66 Haoyu Zhang and Qin Zhang. Minsearch: An efficient algorithm for similarity search under edit distance. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 566–576. ACM, 2020. doi:10.1145/3394486.3403099.
- 67 Zhenjie Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, and Divesh Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In Ahmed K. Elmagarmid and Divyakant Agrawal, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 915–926. ACM, 2010. doi:10.1145/1807167.1807266.
- 68 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Improved design and analysis of practical minimizers. *Bioinform.*, 36(Supplement-1):i119–i127, 2020. doi:10.1093/bioinformatics/btaa472.
- 69 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Lower density selection schemes via small universal hitting sets with short remaining path length. In Russell Schwartz, editor, *Research in Computational Molecular Biology - 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10-13, 2020, Proceedings*, volume 12074 of *Lecture Notes in Computer Science*, pages 202–217. Springer, 2020. doi:10.1007/978-3-030-45257-5_13.
- 70 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Sequence-specific minimizers via polar sets. *bioRxiv*, 2021. doi:10.1101/2021.02.01.429246.

The Visibility Center of a Simple Polygon

Anna Lubiw ✉

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Anurag Murty Naredla ✉

David R. Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

We introduce the *visibility center* of a set of points inside a polygon – a point c_V such that the maximum geodesic distance from c_V to see any point in the set is minimized. For a simple polygon of n vertices and a set of m points inside it, we give an $O((n + m) \log(n + m))$ time algorithm to find the visibility center. We find the visibility center of *all* points in a simple polygon in $O(n \log n)$ time.

Our algorithm reduces the visibility center problem to the problem of finding the geodesic center of a set of half-polygons inside a polygon, which is of independent interest. We give an $O((n + k) \log(n + k))$ time algorithm for this problem, where k is the number of half-polygons.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Visibility, Shortest Paths, Simple Polygons, Facility Location

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.65

Related Version *Full Version:* <https://arxiv.org/abs/2108.07366>

Funding *Anna Lubiw:* Supported by NSERC.

1 Introduction

Suppose you want to guard a polygon and you have many sensors but only one guard to check on the sensors. The guard must be positioned at a point c_V in the polygon such that when a sensor at any query point u sends an alarm, the guard travels from c_V on a shortest path inside the polygon to *see* point u ; the goal is to minimize the maximum distance the guard must travel. More precisely, we must choose c_V to minimize the maximum, over points u , of the geodesic distance from c_V to a point that sees u . The optimum guard position c_V is called the *visibility center* of the set U of possible query points. See Figure 1. We give an $O((n + m) \log(n + m))$ time algorithm to find the visibility center of a set U of size m in an n -vertex simple polygon. To find the visibility center of *all* points inside a simple polygon, we can restrict our attention to the vertices of the polygon, which yields an $O(n \log n)$ time algorithm.

To the best of our knowledge, the idea of visibility centers is new, though it is a very natural concept that combines two significant branches of computational geometry: visibility problems [12]; and center problems and farthest Voronoi diagrams [5].

There is a long history of finding “center points”, for various definitions of “center”. The most famous of these is Megiddo’s linear time algorithm [20] to find the center of a set of points in the plane (Sylvester’s “smallest circle” problem).

Inside a polygon the relevant distance measure is not the Euclidean distance but rather the shortest path, or *geodesic*, distance. The *geodesic center* of a simple polygon is a point p that minimizes the maximum geodesic distance from p to any point q of the polygon, or equivalently, the maximum geodesic distance from p to any vertex of the polygon. Pollack, Sharir, and Rote [24] gave an $O(n \log n)$ time divide-and-conquer algorithm to find the geodesic center of a polygon. Our algorithm builds on theirs. A more recent algorithm by



© Anna Lubiw and Anurag Murty Naredla;
licensed under Creative Commons License CC-BY 4.0

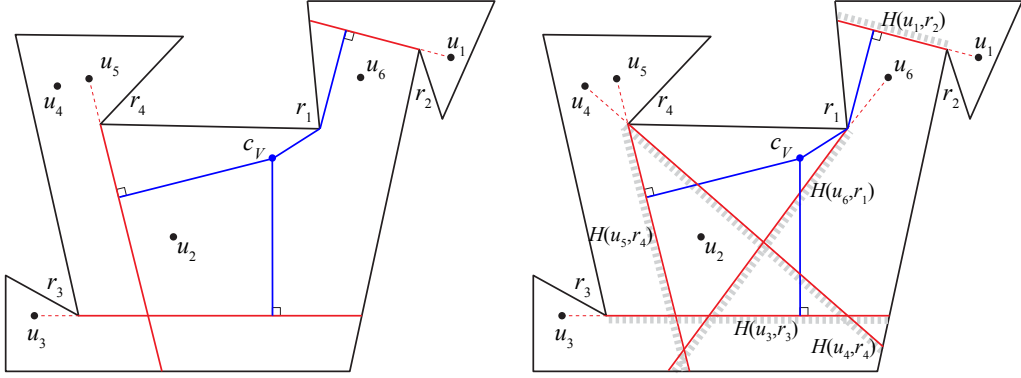
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 65; pp. 65:1–65:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** (left) Point c_V is the *visibility center* of points $U = \{u_1, \dots, u_6\}$. Starting from c_V , the three points we need to travel (equally) farthest to see are u_1, u_3 and u_5 . The shortest paths (in blue) to see these points must reach the half-polygons bounded by the chords (in red) emanating from the points. (right) Equivalently, c_V is the geodesic center of five half-polygons (each shown as a red boundary chord shaded on one side).

Ahn et al. [1] finds the geodesic center of a polygon in linear time. Another notion of the center of a polygon is the link center, which can be found in $O(n \log n)$ time [10].

Center problems are closely related to farthest Voronoi diagrams, since the center is (modulo degeneracies) a vertex of the corresponding farthest Voronoi diagram. Finding the farthest Voronoi diagram of points in the plane takes $\Theta(n \log n)$ time – thus is it strictly harder to find the farthest Voronoi diagram than to find the center. However, working inside a simple polygon helps for farthest Voronoi diagrams: the farthest geodesic Voronoi diagram of the vertices of a polygon can be found in time $O(n \log \log n)$ [23]. Generalizing the two scenarios (points in the plane, and polygon vertices), yields the problem of finding the farthest Voronoi diagram of m points in a polygon, which was first solved by Aronov et al. [3] with run-time $O((n+m) \log(n+m))$, and improved in a sequence of papers [23, 6, 22], with the current best run-time of $O(n + m \log m)$ [27].

Turning to visibility problems in a polygon, there are algorithms for the “quickest visibility problem” – to find the shortest path from point s to see point q , and to solve the query version where s is fixed and q is a query point [2, 26]. For a simple polygon [2], the preprocessing time and space are $O(n)$ and the query time is $O(\log n)$. We did not find these results useful in our algorithm to find the visibility center c_V , but they are useful afterwards to find the actual shortest path from c_V to see a query point.

A more basic version of our problem is to find, if there is one, a point that sees all points in U . The set of such points is the *kernel* of U . When U is the set of vertices, the kernel can be found in linear time [19]. For a general set U , Ke and O’Rourke [18] gave an $O(n + m \log(n+m))$ time algorithm to compute the kernel, and we use some of their results in our algorithm.

Another problem somewhat similar to ours is the watchman problem [9, 11] – to find a minimum length tour from which a single guard can see the whole polygon. Our first step is similar in flavour to the first step for the watchman problem, namely, to replace the condition of “seeing” everything by a condition of visiting certain “essential chords”.

Our Results

The *distance to visibility* from a point x to point u in P , denoted $d_V(x, u)$ is the minimum distance in P from x to a point y such that y sees u . For a set of points U in P , the *visibility*

radius of x with respect to U is $r_V(x, U) := \max\{d_V(x, u) : u \in U\}$. The *visibility center* c_V of U is a point x that minimizes $r_V(x, U)$. Our main result is:

► **Theorem 1.** *There is an algorithm to find the visibility center of a point set U in a simple n -vertex polygon P with run-time $O((n + m) \log(n + m))$ where m is the size of U .*

The key to our algorithm is to reformulate the visibility center problem in terms of distances to certain *half-polygons* inside the polygon. We illustrate the idea by means of the example in Figure 1 where the visibility center of the 6-element point set U is the *geodesic center* of a set of five half-polygons.

More generally, we will reduce the problem of finding the visibility center to the problem of finding a geodesic center of a linear number of half-polygons. The input to this problem is a set \mathcal{H} of k half-polygons (see Section 2 for precise definitions) and the goal is to find a *geodesic center* c that minimizes the maximum distance from c to a half-polygon. More precisely, the *geodesic radius* from a point x to \mathcal{H} is $r(x, \mathcal{H}) := \max\{d(x, H) : H \in \mathcal{H}\}$, and the *geodesic center* c of \mathcal{H} is a point x that minimizes $r(x, \mathcal{H})$. Our second main result is:

► **Theorem 2.** *There is an algorithm to find the geodesic center of a set \mathcal{H} of half-polygons in a simple n -vertex polygon P with run-time $O((n + k) \log(n + k))$ where k is the size of \mathcal{H} .*

Our algorithm extends the divide-and-conquer approach that Pollack et al. [24] used to compute the geodesic center of the vertices of a simple polygon.

Our main motivation for finding the geodesic center of half-polygons is to find the visibility center, but the geodesic center of half-polygons is of independent interest. Euclidean problems of a similar flavour are to find the center (or the farthest Voronoi diagram) of line segments or convex polygons in the plane [7, 17]. These problems are less well-studied than the case of point sites (e.g., see [4] for remarks on this). The literature for geodesic centers is even more sparse, focusing almost exclusively on geodesic centers of points in a polygon. It is thus interesting that the center of half-polygons inside a polygon can be found efficiently. As a special case, we can find the geodesic center of the edges of a simple polygon in $O(n \log n)$ time.

The reduction from the visibility center problem to the geodesic center of half-polygons is in Section 3. The run time is $O((n + m) \log(n + m))$. The algorithm that proves Theorem 2 is in Section 4. Together these prove Theorem 1.

2 Preliminaries

We add a few more basic definitions to augment the main definitions given above. We work with a simple polygon P of n vertices whose boundary ∂P is directed clockwise. A *chord* of P is a line segment inside P that intersects ∂P only at its two endpoints. Any chord divides P into two *half-polygons*. A half-polygon is specified by its chord (p, q) with the convention that the half-polygon contains the path clockwise from p to q .

The *geodesic distance* $d(x, y)$ (or simply, *distance*) between two points x and y in P is the length of the shortest path $\pi(x, y)$ in P from x to y . For half-polygon H , the *geodesic distance* $d(x, H)$ is the minimum distance from x to a point in H .

Points x and y in P are *visible* (x “sees” y) if the segment xy lies inside P . The *distance to visibility* from x to u , denoted $d_V(x, u)$ is the minimum distance from x to a point y such that y sees u . If x sees u , then this distance is 0, and otherwise it is the distance from x to the half-polygon defined as follows. Let r be the last reflex vertex on the shortest path from x to u . Extend the ray \vec{ur} from r until it hits the polygon boundary ∂P at a point p to

obtain a chord rp (which is an edge of the *visibility polygon* of u). Of the two half-polygons defined by rp , let $H(u, r)$ be the one that contains u . See Figure 1.

► **Observation 3.** $d_V(x, u) = d(x, H(u, r))$.

At their core, our methods depend on convexity properties of the distance functions. A basic result is the following which is proved in the full version of our paper.

▷ **Claim 4.** The set of points x with $r_V(x, U) \leq k$ [or with $r(x, \mathcal{H}) \leq k$] is *geodesically convex*, i.e., if two points x, y lie in the set then so does $\pi(x, y)$.

More detail on convexity properties can be found in the long version of our paper. These convexity properties allow us to prove that the visibility center of a set of points U and the geodesic center of a set of half-polygons \mathcal{H} are unique except in very special cases. We explain this for the geodesic center of half-polygons, but the same argument works for the visibility center (or, alternatively, one can use the reduction from the visibility center to the geodesic center in Section 3). First of all, if the geodesic radius is 0 then any point in the intersection of the half-polygons is a geodesic center. So we assume that the geodesic radius r is positive. Then we have the following (proved in the long version):

▷ **Claim 5.** There is a set $\mathcal{H}' \subseteq \mathcal{H}$ of two or three half-polygons such that the set of geodesic centers of \mathcal{H} is equal to the set of geodesic centers of \mathcal{H}' and furthermore

1. if \mathcal{H}' has size 3 then the geodesic center is unique (see Figure 1)
2. if \mathcal{H}' has size 2 then either the geodesic center is unique or the two half-polygons of \mathcal{H}' have chords that are parallel and the geodesic center consists of a line segment parallel to them and midway between them.

3 Reducing the Visibility Center to the Center of Half-Polygons

In this section we reduce the problem of finding the visibility center of a set of points U in a polygon P to the problem of finding the geodesic center of a linear number of “essential” half-polygons \mathcal{H} , which is solved in Section 4.

By Observation 3 (and see Figure 1) the visibility center of U is the geodesic center of the set of $O(mn)$ half-polygons $H(u, r)$ where $u \in U$, r is a reflex vertex of P that sees u , and $H(u, r)$ is the half-polygon containing u and bounded by the chord that extends \vec{ur} from r until it hits ∂P at a point t . Note that finding t is a ray shooting problem and costs $O(\log n)$ time after an $O(n)$ time preprocessing step [16].

However, this set of half-polygons is too large. We will find a set \mathcal{H} of $O(n)$ “essential” half-polygons that suffice, i.e., such that the visibility center of U is the geodesic center of the half polygons of \mathcal{H} . In fact, we give two possible sets of essential half-polygons, $\mathcal{H}_{\text{reflex}}$ and $\mathcal{H}_{\text{hull}}$, where the latter set can be found more efficiently. The bottleneck is still the algorithm for geodesic center of half-polygons, but it still seems worthwhile to optimize the reduction.

We first observe that any half-polygon that contains another one is redundant. For example, in Figure 1 $H(u_4, r_4)$ is redundant because it contains $H(u_5, r_4)$. At each reflex vertex r of P , there are at most two minimal half-polygons $H(u, r)$. Define $\mathcal{H}_{\text{reflex}}$ to be this set of minimal half-polygons. Note that $\mathcal{H}_{\text{reflex}}$ has size $O(n_r)$ where n_r is the number of reflex vertices of P .

Observe that for the case of finding the visibility center of *all* points of P , $\mathcal{H}_{\text{reflex}}$ consists of the half-polygons $H(v, r)$ where (v, r) is an edge of P , so $\mathcal{H}_{\text{reflex}}$ can be found in time $O(n + n_r \log n)$.

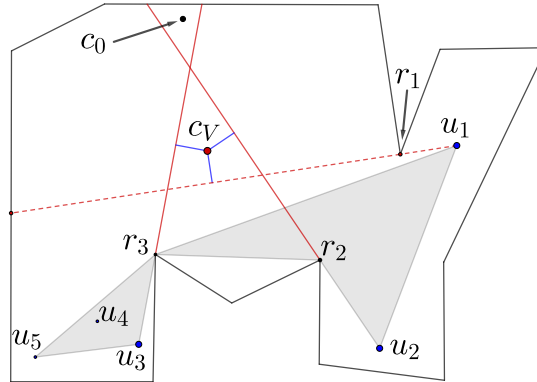
For a point set U , the set $\mathcal{H}_{\text{reflex}}$ was also used by Ke and O'Rourke [18] in their algorithm to compute the kernel of point set U in polygon P . (Recall from the Introduction that the kernel of U is the set of points in P that see all points of U .) They gave a sweep line algorithm ("Algorithm 2") to find $\mathcal{H}_{\text{reflex}}$ in time $O((n+m)\log(n+m))$. To summarize:

► **Proposition 6.** *The geodesic center of $\mathcal{H}_{\text{reflex}}$ is the visibility center of U . Furthermore, $\mathcal{H}_{\text{reflex}}$ can be found in time $O((n+m)\log(n+m))$.*

In the remainder of this section we present a second approach using $\mathcal{H}_{\text{hull}}$ that eliminates the $O(n\log n)$ term. This does not change the runtime to find the visibility center, but it means that improving the algorithm to find the geodesic center of half-polygons will automatically improve the visibility center algorithm. The idea is that $\mathcal{H}_{\text{reflex}}$ is wasteful in that a single point $u \in U$ can give rise to n_r half-polygons. Note that we really only need three half-polygons in an essential set, though the trouble is to find them!

We first eliminate the case where the kernel of U is non-empty (i.e., $r_V = 0$) by running the $O(n+m\log(n+m))$ time kernel-finding algorithm of Ke and O'Rourke [18]. Next we find $\mathcal{H}_{\text{hull}}$ in two steps. First make a subset \mathcal{H}_0 as follows. Construct R , the geodesic convex hull of U in P in time $O(n+m\log(m+n))$ [14, 25]. For each edge (u, r) of R where $u \in U$ and r is a reflex vertex of P , put $H(u, r)$ into \mathcal{H}_0 . Note that \mathcal{H}_0 has size $O(\min\{n_r, m\})$ so ray shooting to find the endpoints of the chords $H(u, r)$ takes time $O(n + \min\{n_r, m\}\log n)$. Unfortunately, as shown in Figure 2, \mathcal{H}_0 can miss an essential half-polygon.

Next, construct a geodesic center c_0 of \mathcal{H}_0 using the algorithm of Section 4. (Note that the geodesic center can be non-unique and in such cases c_0 denotes any one point from the set of geodesic centers.) Then repeat the above step for $U \cup \{c_0\}$, more precisely, construct R' , the geodesic convex hull of $U \cup \{c_0\}$ in P and for each edge (u, r) of R' where $u \in U$ and r is a reflex vertex of P , add $H(u, r)$ to \mathcal{H}_0 . This defines $\mathcal{H}_{\text{hull}}$. Again, $\mathcal{H}_{\text{hull}}$ has size $O(\min\{n_r, m\})$ and ray shooting costs $O(n + \min\{n_r, m\}\log n)$.



■ **Figure 2** The geodesic convex hull of $U = \{u_1, \dots, u_5\}$ is shaded grey. \mathcal{H}_0 consists of the two half-polygons $H(u_2, r_2)$ and $H(u_3, r_3)$ (with solid red chords), but misses $H(u_1, r_1)$, which is essential for the visibility center c_V . The point c_0 denotes a geodesic center of \mathcal{H}_0 .

► **Theorem 7.** *Suppose the kernel of U is empty. Then the geodesic center of $\mathcal{H}_{\text{hull}}$ is the visibility center of U . Furthermore $\mathcal{H}_{\text{hull}}$ can be found in time $O(n+m\log(n+m))$ plus the time to find the geodesic center of $O(\min\{n_r, m\})$ half-polygons.*

Proof. The run-time was analyzed above. Consider the visibility center c_V . By assumption, $r_V > 0$. We consider the half-polygons $H(u, r) \in \mathcal{H}_{\text{reflex}}$ such that $r_V = d(c_V, H(u, r))$.

By Claim 5 either there are three of these half-polygons, H_1 , H_2 and H_3 , that uniquely determine c_V , or there are two, H_1 and H_2 , that determine c_V . Then c_V is the geodesic center of H_i $i = 1, 2, 3$ or $i = 1, 2$ depending on which case we are in. Let $H_i = H(u_i, r_i)$.

If all the H_i 's are in \mathcal{H}_0 , we are done. We will show that at least two are in \mathcal{H}_0 and the third one (if it exists) is “caught” by c_0 . See Figure 2. Let h_i be the chord defining H_i and let \overline{H}_i be the other half-polygon determined by h_i .

▷ **Claim 8.** If U contains a point in \overline{H}_i then (u_i, r_i) is an edge of R so $H_i \in \mathcal{H}_0$.

Proof. Let u be a point in \overline{H}_i . Observe that $\pi(u_i, u)$ contains the segment $u_i r_i$. Thus r_i is a vertex of R . Furthermore $u_i r_i$ is an edge of R . (Note that H_i is extreme at r since we picked it from $\mathcal{H}_{\text{reflex}}$.) Thus H_i is in \mathcal{H}_0 . ◁

▷ **Claim 9.** At least two of the H_i 's lie in \mathcal{H}_0 .

Proof. First observe that if two of the half-polygons are disjoint, say H_i and H_j , then they lie in \mathcal{H}_0 , because $u_i \in H_i$ implies $u_i \in \overline{H}_j$ so by Claim 8, $H_i \in \mathcal{H}_0$, and symmetrically, $H_j \in \mathcal{H}_0$.

We separate the proof into cases depending on the number of H_i 's. If there are two then they must be disjoint otherwise a point in their intersection would be a visibility center with visibility radius $r_V = 0$. Then by the above observation, they are both in \mathcal{H}_0 .

It remains to consider the case of three half-polygons. If two are disjoint, we are done, so suppose each pair H_i, H_j intersects. Then the three chords h_i form a triangle. Furthermore, since $\bigcap \overline{H}_i$ is non-empty (it contains c_V), the inside of the triangle is $\bigcap \overline{H}_i$. Now suppose $H_1 \notin \mathcal{H}_0$. Then by Claim 8, $u_2, u_3 \in H_1$. This implies (see Figure 2) that $u_2 \in \overline{H}_3$ and $u_3 \in \overline{H}_2$, so by Claim 8, H_2 and H_3 are in \mathcal{H}_0 . ◁

We now complete the proof of the theorem. We only need to consider the case of three H_i 's, where one of them, say H_1 , is not in \mathcal{H}_0 . Our goal is to show that c_0 , the geodesic center of \mathcal{H}_0 , lies in \overline{H}_1 and thus H_1 is in $\mathcal{H}_{\text{hull}}$. Let $X = \{x \in P : d(x, H_2) \leq r_V \text{ and } d(x, H_3) \leq r_V\}$. Observe that $c_0 \in X$ (because the radius is non-increasing as we eliminate half-polygons). Now, c_V is the unique point within distance r_V of the half-polygons H_1, H_2 and H_3 . If $c_0 \in H_1$, then c_0 's distance to H_1 would be 0 which contradicts the uniqueness property of c_V . Thus $c_0 \in \overline{H}_1$. By the same reasoning as in Claim 8, this implies that $u_1 r_1$ is an edge of R' , the geodesic convex hull of $U \cup \{c_0\}$. Thus H_1 is in $\mathcal{H}_{\text{hull}}$ by definition of $\mathcal{H}_{\text{hull}}$. ◀

4 The Geodesic Center of Half-Polygons

In this section, we give an algorithm to find the geodesic center of a set \mathcal{H} of k half-polygons inside an n -vertex polygon P . We preprocess by sorting the half-polygons in cyclic order of their first endpoints around ∂P in time $O(k \log k)$. We assume that no half-polygon in \mathcal{H} contains another – any non-minimal half-polygon is irrelevant and can be discarded. Note that the minimal half-polygons can be found in linear time from the sorted order.

We follow the approach that Pollack et al. [24] used to find the geodesic center of the vertices of a polygon. Many steps of their algorithm rely, in turn, on search algorithms of Megiddo's [20].

The main ingredient of the algorithm is a linear time *chord oracle* that, given a chord $K = ab$ of the polygon, finds the *relative geodesic center*, c_K (the optimum center point restricted to points on the chord), and tells us which side of the chord contains the center. We must completely redo the chord oracle in order to handle paths to half-polygons instead of vertices, but the main steps are the same. Our chord oracle runs in time $O(n + k)$. Pollack

et al.'s chord oracle was used as a black box in subsequent faster algorithms [1], so we imagine that our version will be an ingredient in any faster algorithm for the geodesic center of half-polygons.

Using the chord oracle, we again follow the approach of Pollack et al. to find the geodesic center. The total run time is $O((n + k) \log(n + k))$.

We give a road-map for the remainder of this section, listing the main steps, which are the same as those of Pollack et al., and highlighting the parts that we must rework.

§ 4.1 A Linear Time Chord Oracle

1. Test a candidate center point. Given a point x on the chord $K = ab$, is the relative geodesic center c_K to the left or right of x ? Is the geodesic center c to the left or right of chord K ?
2. Find shortest paths from a and from b to all half-polygons. The details of this step are novel, because we need shortest paths to half-polygons rather than vertices.
3. Find a linear number of simple functions defined on K whose upper envelope is the geodesic radius function. We must redo this from the ground up.
4. Find the relative center on K (the point that minimizes the geodesic radius function) using Megiddo's technique.

§ 4.2 Finding the Geodesic Center of Half-Polygons

1. Use the chord oracle to find a region of P that contains the center and such that for any half-polygon $H \in \mathcal{H}$, all geodesic paths from the region to H are combinatorially the same. We give a more modern version of this step using epsilon nets.
2. Solve the resulting Euclidean “intersection radius problem” – to find a smallest disk that contains given disks and intersects given lines. This is new because of the condition about intersecting given lines.

4.1 A Linear Time Chord Oracle

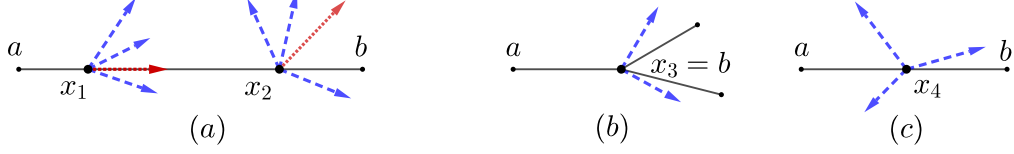
In this section we give a linear time chord oracle. Given a chord $K = ab$ the chord oracle tells us whether the geodesic center of \mathcal{H} lies to the left, to the right, or on the chord K . It does this by first finding the relative geodesic center $c_K = \operatorname{argmin}\{r(x, \mathcal{H}) : x \in K\}$, together with the half-polygons that are farthest from c_K . From this information, we can identify which side of K contains the geodesic center c in the same way as Pollack et al. by testing the vectors of the first segments of the shortest paths from c_K to its furthest half-polygons. This test is described in Subsection 4.1.1.

The chord oracle thus reduces to the problem of finding the relative geodesic center and its farthest half-polygons. The main idea here is to capture the geodesic radius function along the chord (i.e., the function $r(x, \mathcal{H})$ for $x \in K$) as the upper envelope of a linear number of simple functions defined on overlapping subintervals of K . In order to find the simple functions (Section 4.1.3) we first compute shortest paths from a and from b to all the half-polygons (Section 4.1.2). Finally we apply Megiddo's techniques (Section 4.1.4) to find the point c_K on K that minimizes the geodesic radius function.

4.1.1 Testing a Candidate Center Point

Given the relative geodesic center c_K on chord K and the first edges of the paths from c_K to its farthest half-polygons, we can test in constant time whether the geodesic center is equal to c_K or lies to the left or the right of K . We can also test, given a point x on K and the

first edges of the paths to its farthest half-polygons, whether c_K is equal to x or lies to the left or right of x . We illustrate the tests in Figure 3, and defer a more rigorous explanation to the long version of our paper.



■ **Figure 3** Points x_i on chord $K = ab$ with directions of paths to farthest half-polygons in dashed blue and the direction for improvement in dotted red. (a) x_1 is not a relative center. x_2 is a relative center but not the true geodesic center. (b,c) x_3 and x_4 are geodesic centers.

4.1.2 Shortest Paths to Half-Polygons

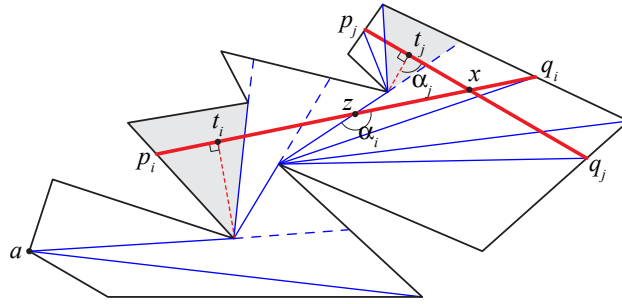
In this section we give a linear time algorithm to find the shortest path tree from point a on the polygon boundary to all the half-polygons \mathcal{H} . Recall that each half-polygon is specified by an ordered pair of endpoints on ∂P , and the half-polygons are sorted in clockwise cyclic order by their first endpoints. From this, we identify the half-polygons that contain a , and we discard them – their distance from a is 0. Let $H_1, \dots, H_{k'}$ be the remaining half-polygons where H_i is bounded by endpoints $p_i q_i$, and the H_i 's are sorted by p_i , starting at a .

The idea is to first find the shortest path map T_a from a to the set consisting of the polygon vertices and the points p_i and q_i . Recall that the shortest path map is an augmentation of the shortest path tree that partitions the polygon into triangular regions in which the shortest path from a is combinatorially the same (see Figure 4). The shortest path map can be found in linear time [13]. Note that T_a is embedded in the plane (none of its edges cross) and the ordering of its leaves matches their ordering on ∂P . Our algorithm will traverse T_a in depth-first order, and visit the triangular regions along the way.

Our plan is to augment T_a to a shortest path tree \bar{T}_a that includes the shortest paths from a to each half-polygon H_i . Note that \bar{T}_a is again an embedded ordered tree. We can find $\pi(a, H_i)$ by examining the regions of the shortest path map intersected by $p_i q_i$. These lie in the *funnel* between the shortest paths $\pi(a, p_i)$ and $\pi(a, q_i)$. Note that edges of the shortest path map T_a may cross the chord $p_i q_i$. Also, the funnels for different half-polygons may overlap. The key to making the search efficient is the following lemma:

► **Lemma 10.** *The ordering $H_1, H_2, \dots, H_{k'}$ matches the ordering of the paths $\pi(a, H_i)$ in the tree \bar{T}_a .*

Proof. Consider two half-polygons $H_i = p_i q_i$ and $H_j = p_j q_j$, with $i < j$. We prove that $\pi(a, H_i)$ comes before $\pi(a, H_j)$ in \bar{T}_a . If H_i and H_j are disjoint, the result is immediate since the corresponding funnels do not overlap. Otherwise (because neither half-polygon is contained in the other) $p_i q_i$ and $p_j q_j$ must intersect, say at point x . See Figure 4. Let t_i and t_j be the terminal points of the paths $\pi(a, H_i)$ and $\pi(a, H_j)$, respectively. If t_i lies in $p_i x$ and t_j lies in $x q_j$ then the result follows since t_i and t_j lie in order on the boundary of the truncated polygon formed by removing H_i and H_j . So suppose that t_j lies in $p_j x$ (the other case is symmetric). Then $\pi(a, t_j)$ crosses $p_i q_i$ at a point z in $p_i x$. From z to t_j the path $\pi(a, t_j)$ lies inside the cone with apex x bounded by the rays from x through z and from x through t_j . Within that cone, the path only turns left. The angle α_j at t_j is $\geq 90^\circ$ (it may be $> 90^\circ$ if $t_j = p_j$), which implies that the angle α_i at z is $> 90^\circ$. Therefore t_i lies to the left of z , as required. ◀



■ **Figure 4** The shortest path map T_a (thin blue) and the augmentation (dashed red) to include shortest paths to the two half-polygons bounded by chords $p_i q_i$ and $p_j q_j$ (thick red).

Based on the Lemma, the algorithm traverses the regions of the shortest path map T_a in depth first search order, and traverses the half-polygons H_i in order $i = 1, 2, \dots, k'$. It is easy to test if one region contains the shortest path to H_i (either to p_i , or to q_i , or reaching an internal point of $p_i q_i$ at a right angle); if it does, we increment i , and otherwise we proceed to the next region. The total time is $O(n + k)$.

4.1.3 Functions to Capture the Distance to Farthest Half-Polygons

In this section we give a linear time algorithm to find a linear number of simple functions defined on the chord $K = ab$ whose upper envelope is the geodesic radius function $r(x, \mathcal{H})$ for $x \in K$. Specifically, we use the shortest path trees \bar{T}_a and \bar{T}_b constructed in the previous section to build a set of $O(n + k)$ pairs f, I where:

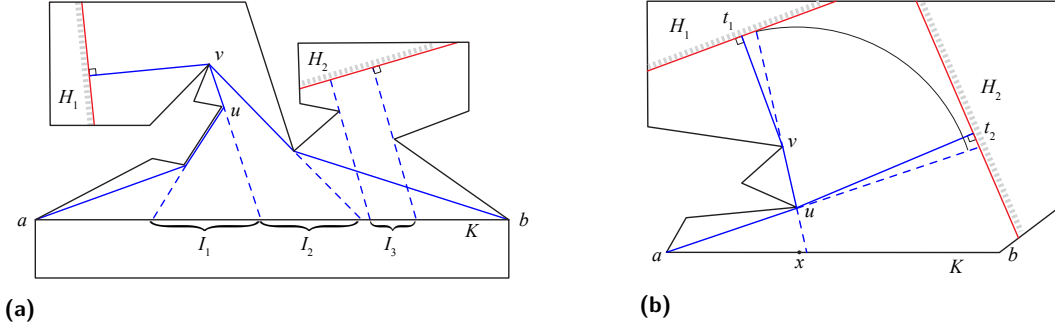
- I is a subinterval of K and f is a function defined on domain I ,
- each function has the form $f(x) = d_2(x, s) + \kappa$ where κ is a constant, d_2 is Euclidean distance, and s is a point or a line,
- for any point $x \in K$ the maximum of $f(x)$ over intervals I containing x is equal to $r(x, \mathcal{H})$.

For intuition, see Figure 5a, which shows several intervals and their associated functions. Note that we deal separately with the two pieces of the polygon on each side of K . There is an obvious set of $O(nk)$ pairs f, I with the above properties, one for each $H \in \mathcal{H}$ and $v \in P$, but we want a set of linear size.

The crucial property is that there are a limited number of farthest half-polygons associated with each vertex, and we can restrict our attention (mostly) to longest paths in the trees \bar{T}_a and \bar{T}_b . In particular, consider a point $x \in K$, a half-polygon H , and the shortest path $\pi(x, H)$. Suppose $\pi(x, H)$ has at least two edges, say $\pi(x, H) = x, u, v, \dots, t$, where t is the terminal point on H . If $\pi(x, H)$ turns left at u , then the part of the path from u to t is part of the tree \bar{T}_a , and symmetrically for right turns and \bar{T}_b . See Figure 5b. Furthermore, if H is a farthest half-polygon from x , then we can show (in the full version) that the part of the path from v to t is the longest path in \bar{T}_a descending from v to any half-polygon. Note that using longest paths from u rather than v is not correct – see Figure 5b.

For any node u in \bar{T}_a let $\ell_a(u)$ be the maximum length of a path in \bar{T}_a descending from u to a node representing a terminal point on some half-polygon. Define $\ell_b(u)$ similarly. We can compute these functions in linear time in leaf-to-root order. In the situation described above, where H is a farthest half-polygon from x and the path $\pi(x, H) = x, u, v, \dots, t$ turns left at u , we have $d(x, H) = |xu| + |uv| + \ell_a(v)$, which is a simple function of the desired form.

The algorithm will output these functions for all pairs u, v where the following conditions hold: u is a vertex visible from a point on the chord ab (equivalently, u has different parents



■ **Figure 5** (a) An illustration of functions and intervals. For x in interval I_1 , $d(x, H_1) = d_2(x, u) + \kappa_1$. For x in I_2 , $d(x, H_1) = d_2(x, v) + \kappa_2$. For x in I_3 , $d(x, H_2) = d_2(x, H_2)$. (b) From point x the farthest half-polygon is H_1 via the path x, u, v, t_1 . This matches the longest path in \bar{T}_a descending from v (which goes to H_1) but does not match the longest path descending from u (which goes to H_2).

in \bar{T}_a and \bar{T}_b); v is a child of u in one of the trees; and x lies in an appropriate interval on ab such that u, v can be the start of a geodesic path from x . We deal separately with shortest paths that go from a point $x \in ab$ to a half-polygon without going through any vertices. Observe that the number of such functions is $O(n + k)$.

We defer further details of the algorithm to the long version of our paper. Besides enhancing the method of Pollack et al. [24] to deal with half-polygons, our aim is to give a clearer and easier-to-verify presentation.

4.1.4 Finding the Relative Geodesic Center on a Chord

The last step of the chord oracle is exactly the same as in Pollack et al. [24]. Given the set of $O(n + k)$ simple convex functions whose upper envelope is the geodesic radius function on chord K (from Section 4.1.3), and given the test of whether the relative center is left/right of a point on K (from Section 4.1.1) we want to find the relative center, c_K , that minimizes the radius function. Pollack et al. use a technique of Megiddo's to do this in $O(n + k)$ time. The idea is to pair up the functions, find the intersection and domain end-points of each pair, and test medians of those in order to eliminate a constant fraction of the functions in each round. Further details are in the long version of this paper. Finally, to find the paths from the relative center c_K to its farthest half-polygons, use the linear time shortest path algorithm (Section 4.1.2) on each side of K .

4.2 Finding the Geodesic Center of Half-Polygons

In this section we show how to use the $O(n + k)$ time chord oracle from Section 4.1 to find the geodesic center of the k half-polygons in $O((n + k) \log(n + k))$ time. The basic structure of the algorithm is the same as that of Pollack et al. [24].

In the first step we use the chord oracle to restrict the search for the geodesic center to a small region where the problem reduces to a Euclidean “intersection radius problem”. In the second step we solve the resulting problem, which involves some new ingredients to handle our case of half-polygons. Each step takes $O((n + k) \log(n + k))$ time.

4.2.1 Finding a Region that Contains the Geodesic Center

Triangulate P in linear time [8]. Run the chord oracle on a chord of the triangulation that splits the polygon into balanced pieces and recurse on the appropriate subpolygon. In $O(\log n)$ iterations, we narrow our search down to one triangle T^* of the triangulation. This step takes $O((n+k)\log n)$ time.

Next, we refine T^* to a region R that contains the center and such that R is *homogeneous*, meaning that for any $H \in \mathcal{H}$ the shortest paths from points in R to H have the same combinatorial structure (the same sequence of polygon vertices along the path).

The idea is to subdivide T^* by $O(n+k)$ lines so that each cell in the resulting line arrangement is homogeneous, and then to find the cell containing the center. Construct the shortest path trees to \mathcal{H} from each of the three corners of triangle $T^* = (a^*, b^*, c^*)$ using the algorithm of Section 4.1.2. For each edge (u, v) of each tree, add the line through uv if it intersects T^* . (In fact, we do not need all these lines. The situation is similar to that shown in Figure 5a, with a^*, b^* in place of a, b . We need the dashed lines shown in the figure. In particular, it suffices to use tree edges (u, v) such that u is visible from an edge of T^* .)

We add three more lines for each half-polygon $H \in \mathcal{H}$, specifically, the chord h that defines H , and the two lines perpendicular to h through the endpoints of h .

The result is a set of $O(n+k)$ lines that we obtain in time $O(n+k)$. It is easy to prove that the resulting line arrangement has homogeneous regions.

All that remains is to find the cell of the arrangement that contains the geodesic center. It is simpler to state the algorithm in terms of ϵ -nets instead of the rather involved description of Megiddo's technique used by Pollack et al. [24]. Informally, to find a homogeneous region, we will look at a range space on ground set L whose ranges consist of the subsets of L that intersect some triangle. Such a range space can be shown to have constant sized ϵ -nets [15]. By using the $O(n+k)$ time chord oracle a constant number of times (on a constant sized ϵ -net) we can restrict the search space to a region that intersects only a fraction of the original lines. Repeating this step for $O(\log(n+k))$ times, we arrive at a region R with the required properties. The details are deferred to the long version due to space constraints. The total runtime for this step is $O((n+k)\log(n+k))$.

4.2.2 Solving an Unconstrained Problem

At this point, we know that the polygonal region R contains a geodesic center of the set \mathcal{H} of half-polygons in P . Furthermore, R is homogeneous. We can pick a point p in R and find the shortest path tree from p to all half-polygons. If $\pi(p, H)$ reaches an internal point of the chord h defining H then $d(x, H) = d_2(x, h)$ for all $x \in R$. And if the first segment of $\pi(x, H)$ reaches a vertex u , then $d(x, H) = d_2(x, u) + \kappa$ for all $x \in R$, where κ is a constant. Our goal is to find the point $x \in R$ that minimizes the maximum over $H \in \mathcal{H}$ of $d(x, H)$. Since this point must lie in the region R (a guarantee we have from the earlier steps), we can completely disregard the underlying polygon P in solving the problem.

In the Euclidean plane, the problem may be reinterpreted in a geometric manner. We wish to find the circle of smallest radius that *contains* $O(n)$ disks and *intersects* $O(k)$ lines. The disk constraints in the new interpretation correspond to half-polygons H where $d(x, H) = d_2(x, u) + \kappa$ – the disk is centered at u with radius κ . The line constraints correspond to half-polygons H where $d(x, H) = d_2(x, h)$.

This is a combination of problems referred to as the spanning circle problem [21] or the intersection radius problem [7] in the literature. We will call it the *intersection radius*

problem for disks and lines, although the name is not entirely accurate. In the long version of this paper we prove:

► **Lemma 11.** *The intersection radius problem for disks and lines can be solved in linear time.*

We outline the method. The problem for disks alone was solved by Megiddo [21] using an ingenious idea. The two-dimensional problem is modified to a problem in three dimensions and the constraints modified in such a way that the bisector between the constraints for two disks becomes a plane in three dimensions. Thereafter, techniques from linear-time algorithms for linear programming are used to prune away disks that do not define the final answer [20]. The prune-and-search technique prunes away a constant fraction of those disks in linear time and repeating the process reduces the number of disks to some constant number, after which a brute force method may be employed.

To extend this to handle our line constraints, we add constraints that ensure that the distances to the lines are less than the radius of the final disk. The bisectors between two lines are an angle bisector pair. These bisectors become vertical planes in the transformed three dimensional version of the problem. We thus have a set of planes in three dimensions that are bisectors between pairs of lines or pairs of disks. Using prune-and-search in two phases per iteration and a few other ideas ([7]), we modify Megiddo's technique for disks to solve our problem in linear time.

5 Conclusions

We introduced the notion of the visibility center of a set of points in a polygon and gave an algorithm with run time $O((n + m) \log(n + m))$ to find the visibility center of m points in an n -vertex polygon. To do this, we gave an algorithm to find the geodesic center of a given set of half-polygons inside a polygon, a problem of independent interest. We conclude with some open questions.

Can the visibility center of a simple polygon be found more efficiently? Note that the geodesic center of the vertices of a simple polygon can be found in linear time [1]. Our current method involves ray tracing and sorting, which are serious barriers. A more reasonable goal is to find the visibility center of m points in a polygon in time $O(n + m \log m)$.

Is there a more efficient algorithm to find the geodesic center of (sorted) half-polygons? In forthcoming work we give a linear time algorithm for the special case of finding the geodesic center of the *edges* of a polygon (this is the case where the half-polygons hug the edges).

How hard is it to find the farthest visibility Voronoi diagram of a polygon? Finally, what about the 2-visibility center of a polygon, where we can deploy two guards instead of one?

References

- 1 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56(4):836–859, 2016. doi:10.1007/s00454-016-9796-0.
- 2 Esther M Arkin, Alon Efrat, Christian Knauer, Joseph S B Mitchell, Valentin Polishchuk, Günter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7:77–100, 2016. URL: <https://jocg.org/index.php/jocg/article/view/3001>.
- 3 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, 1993. doi:10.1145/73393.73417.

- 4 Franz Aurenhammer, Robert L Scot Drysdale, and Hannes Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006. doi:10.1016/j.ipl.2006.07.008.
- 5 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing Company, 2013.
- 6 Luis Barba. Optimal algorithm for geodesic farthest-point Voronoi diagrams. In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. URL: <http://drops.dagstuhl.de/opus/volltexte/2019/10416>.
- 7 Binay K Bhattacharya, Shreesh Jadhav, Asish Mukhopadhyay, and J-M Robert. Optimal algorithms for some intersection radius problems. *Computing*, 52(3):269–279, 1994. doi:10.1007/bf02246508.
- 8 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991. doi:10.1007/bf02574703.
- 9 Wei-Pang Chin and Simeon Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991. doi:10.1007/bf02574671.
- 10 Hristo N Djidjev, Andrzej Lingas, and Jörg-Rüdiger Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8(2):131–152, 1992.
- 11 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph SB Mitchell. Touring a sequence of polygons. In *Proceedings of the thirty-fifth Annual ACM Symposium on Theory of Computing (STOC '03)*, pages 473–482, 2003. doi:10.1145/780542.780612.
- 12 Subir Kumar Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007.
- 13 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987. doi:10.1007/bf01840360.
- 14 Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-x.
- 15 David Haussler and Emo Welzl. ϵ -nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.
- 16 John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.
- 17 Shreesh Jadhav, Asish Mukhopadhyay, and Binay Bhattacharya. An optimal algorithm for the intersection radius of a set of convex polygons. *Journal of Algorithms*, 20(2):244–267, 1996.
- 18 Yan Ke and Joseph O'Rourke. Computing the kernel of a point set in a polygon. In *Workshop on Algorithms and Data Structures*, pages 135–146. Springer, 1989.
- 19 Der-Tsai Lee and Franco P Preparata. An optimal algorithm for finding the kernel of a polygon. *Journal of the ACM (JACM)*, 26(3):415–421, 1979. doi:10.1145/322139.322142.
- 20 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983. doi:10.1137/0212052.
- 21 Nimrod Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(6):605–610, 1989. doi:10.1007/bf02187750.
- 22 Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, 63(2):418–454, 2020.
- 23 Eunjin Oh, Luis Barba, and Hee-Kap Ahn. The geodesic farthest-point Voronoi diagram in a simple polygon. *Algorithmica*, 82(5):1434–1473, 2020. doi:10.1007/s00453-019-00651-z.
- 24 Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989. doi:10.1007/bf02187751.
- 25 Godfried T Toussaint. Computing geodesic properties inside a simple polygon. *Revue d'Intelligence Artificielle*, 3(2):9–42, 1989.

- 26 Haitao Wang. Quickest visibility queries in polygonal domains. *Discrete & Computational Geometry*, 62(2):374–432, 2019. doi:10.1007/s00454-019-00108-8.
- 27 Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point Voronoi diagrams in simple polygons. In *International Symposium on Computational Geometry (SoCG 2021)*, 2021. URL: <https://arxiv.org/pdf/2103.00076.pdf>.

Extension of Additive Valuations to General Valuations on the Existence of EFX

Ryoga Mahara

Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

Envy-freeness is one of the most widely studied notions in fair division. Since envy-free allocations do not always exist when items are indivisible, several relaxations have been considered. Among them, possibly the most compelling concept is envy-freeness up to any item (EFX). We study the existence of EFX allocations for general valuations. The existence of EFX allocations is a major open problem. For general valuations, it is known that an EFX allocation always exists (i) when $n = 2$ or (ii) when all agents have identical valuations, where n is the number of agents. It is also known that an EFX allocation always exists when one can leave at most $n - 1$ items unallocated.

We develop new techniques and extend some results of additive valuations to general valuations on the existence of EFX allocations. We show that an EFX allocation always exists (i) when all agents have one of two general valuations or (ii) when the number of items is at most $n + 3$. We also show that an EFX allocation always exists when one can leave at most $n - 2$ items unallocated. In addition to the positive results, we construct an instance with $n = 3$ in which an existing approach does not work as it is.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Discrete Fair Division, EFX allocations, General Valuations

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.66

Related Version Full Version: <https://arxiv.org/abs/2107.09901>

Acknowledgements The author would like to thank Yusuke Kobayashi for his generous support and useful discussion. This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled “Advanced Mathematical Science for Mobility Society”.

1 Introduction

Fair division of items among competing agents is a fundamental and well-studied problem in Economics and Computer Science. We are given a set M of m items and a set N of n agents with individual preferences. Each agent i has a valuation function $v_i : 2^M \rightarrow \mathbb{R}_{\geq 0}$ for each subset of items. The goal is to distribute items among n agents in a *fair* and *efficient* manner. In this paper, we consider the *indivisible* setting: an item cannot be split among multiple agents. Let an allocation $X = (X_1, X_2, \dots, X_n)$ denote a partition of M into n bundles such that X_i is allocated to agent i . Several concepts of fairness have been considered in the literature, and one of the most well-studied notions of fairness is *envy-freeness*. An allocation X is *envy-free* if for any pair of agents i, j we have $v_i(X_i) \geq v_i(X_j)$, i.e., no agent i envies another agent j ’s bundle. Unfortunately, envy-free allocations do not always exist when items are indivisible. We can easily see this even with two players and a single item having positive value for both of them: one of the agents has to receive the item and the other agent envies her. This motivates the study of relaxations of envy-freeness.

The most compelling relaxations of envy-freeness is *envy-freeness up to any item (EFX)* introduced by Caragiannis et al. [14]. An allocation X is *EFX* if for any pair of agents i, j and for any $g \in X_j$ we have $v_i(X_i) \geq v_i(X_j \setminus \{g\})$, i.e., no agent i envies another agent j after the removal of *any* item in j ’s bundle. EFX is regarded as the best analogue of envy-freeness in discrete fair division: Caragiannis et al. [13] remarked that “*Arguably, EFX is the best fairness analog of envy-freeness for indivisible items.*” However, the existence of



© Ryoga Mahara;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 66; pp. 66:1–66:15

Leibniz International Proceedings in Informatics



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

EFX allocations is known only in a few cases. As described in [14], “*Despite significant effort, we were not able to settle the question of whether an EFX allocation always exists (assuming all goods must be allocated), and leave it as an enigmatic open question.*”

For general valuations, i.e., each valuation function v_i is only assumed to be normalized and monotone, Plaut and Roughgarden [31] showed that an EFX allocation always exists (i) when $n = 2$ or (ii) when all agents have identical valuations. Furthermore, it was shown in [31] that exponentially many value queries may be required to identify EFX allocations even in the case where there are only two agents with identical *submodular* valuation functions. It is not known whether EFX allocations always exist even when $n = 3$ for general valuations.

For additive valuations, i.e., each valuation function v_i is normalized, monotone, and *additive*¹, it is known that an EFX allocation always exists when $n = 3$ [16] or all agents have one of two valuations [30]. It is not known whether EFX allocations always exist even when $n = 4$ for additive valuations.

One of relaxations of EFX is *EFX with charity* introduced by Caragiannis et al. [13]. This is a *partial* EFX allocation, where all items need not be allocated to the agents. Thus some items may be left unallocated. On the other hand, an allocation is said to be *complete* if all items are allocated. For general valuations, Chaudhury et al. [18] showed that there exist a partial EFX allocation and a set of unallocated items U such that no agent envies U , and $|U| \leq n - 1$. For additive valuations, Caragiannis et al. [13] showed that there exists a partial EFX allocation where every agent receives at least half the value of her bundle in an optimal *Nash social welfare* allocation². Quite recently, Berger et al. [8] showed that when $n = 4$, there exists an EFX allocation with at most one unallocated item such that no agent envies the unallocated item. Moreover, they extend their results and existing results in [16] and [30] beyond additive valuations to *nice cancelable* valuations which is a class including additive, unit-demand, budget-additive, multiplicative valuations, and so on.

1.1 Our Contributions

We investigate the existence of EFX allocations for general valuations, i.e., the valuation of each agent is not necessarily additive. To prove the existence of EFX, we iteratively construct an EFX allocation from an existing partial EFX allocation to advance with respect to a certain potential function. Chaudhury et al. [16] introduced the *lexicographic potential function* in order to show that they could advance an allocation while keeping EFX. We use not only the lexicographic potential function but also a new potential function, which we call *partition leximin potential function*. When we construct a new EFX allocation, which is better than the previous one with respect to the potential function, some agent may become worse off than in the previous allocation. The problem is that such an agent may become to envy other agents, which results in violating EFX. Our technical contribution is to develop a new technique to avoid such situations (see Section 1.2).

Using this new technique, we obtain some new results on the existence of EFX. Our results are described below, and are summarized in Table 1. Our first result is for the case where each agent has one of two given valuations. The following theorem extends the case when all agent have the identical valuations [31].

► **Theorem 1.** *There exists a complete EFX allocation when each agent has one of two general valuations.*

¹ A valuation function v_i is additive if $v_i(S) = \sum_{g \in S} v_i(\{g\})$ for any $S \subseteq M$.

² This is an allocation that maximizes $\Pi_{i=1}^n v_i(X_i)$.

It is known that there exists an EFX allocation when each agent has one of two additive valuations [30]. Berger et al. [8] extended this result beyond additive valuations to nice cancelable valuations. In [30] and [8], to prove the result they iteratively construct a Pareto dominating (see Section 2.1) EFX allocation from an existing partial EFX allocation. However, such an approach is not likely to work for general valuations. To prove Theorem 1, we introduce a new potential function (*partition leximin potential function*) and show that for any EFX allocation with at least one unallocated item, one can obtain a new EFX allocation that makes progress with respect to the partition leximin potential function. This implies that there exists a complete EFX allocation. More details are discussed in Section 4.

Our second result concerns EFX with charity. As mentioned above, it is known that there exist a partial EFX allocation and a set of unallocated items U such that no agent envies U , and $|U| \leq n - 1$ [18]. The following theorem strengthens the bound on the number of unallocated items from $n - 1$ to $n - 2$.

► **Theorem 2.** *For general valuations, there exists an EFX allocation X with at most $n - 2$ unallocated items. Moreover, no agent envies the set of unallocated items in X .*

Berger et al. [8] showed that for nice cancelable valuations, there exists an EFX allocation X with at most $n - 2$ unallocated items. Theorem 2 extends this results to general valuations. To prove Theorem 2, we show that for any EFX allocation with at least $n - 1$ unallocated items, one can obtain a new EFX allocation that makes progress with respect to the lexicographic potential function. This implies that there exists an EFX allocation with at most $n - 2$ unallocated items.

We also study the case with a small number of items. For additive valuations, Amanatidis et al. [3] showed that when $m \leq n + 2$, there exists an EFX allocation. For general valuations, to the best of our knowledge, non-trivial results are not known. The following theorem extends the existing results in the sense that it not only increases the number of items, but also makes valuation function general instead of additive.

► **Theorem 3.** *For general valuations, there exists a complete EFX allocation when $m \leq n + 3$.*

To prove Theorem 3, we also use the lexicographic potential function.

In addition to the above positive results, we study a limitation of the approach using the lexicographic potential function. We construct an instance with $n = 3$ and $m = 7$ in which there exists an EFX allocation with one unallocated item such that no progress can be made with respect to the lexicographic potential function. This shows that Theorem 2 and Theorem 3 are the best possibilities in a sence.

■ **Table 1** Our positive EFX results, where $|U|$ is the number of unallocated items.

Setting	Prior results	Our results
EFX for one of two valuations	Additive [30], Nice cancelable [8]	General
EFX with charity	General, $ U \leq n - 1$ [18] Nice cancelable, $ U \leq n - 2$ [8]	General, $ U \leq n - 2$
EFX for a small number of items	Additive, $m \leq n + 2$ [3]	General, $m \leq n + 3$

1.2 Our Techniques

We first fix a potential function ϕ for all allocations. For an existing partial EFX allocation, in order to find a new EFX allocation we use the champion graph introduced in [16]. If we have a Pareto improvable cycle (see Definition 7) in the champion graph, then we can

conclude that there exists an EFX allocation Y Pareto dominating X . That is, we have $v_i(Y_i) \geq v_i(X_i)$ for any agent i , and $v_j(Y_j) > v_j(X_j)$ for some agent j . It would imply that $\phi(X) < \phi(Y)$. Otherwise, it may no longer be possible to Pareto dominate X . Thus, we seek an allocation Y such that some agent i is worse off than in X and every agent other than i is not worse off than in X , i.e., $v_i(Y_i) < v_i(X_i)$ and $v_j(Y_j) \geq v_j(X_j)$ for $j \in N \setminus \{i\}$. We choose such an agent i who is less important with respect to ϕ . In order to preserve EFX, we list bundles that can appear in Y . We then allocate to agent i the best of the bundles in such a list, which is a key ingredient in our construction. Since agent i receives the most favorite bundles in Y , agent i does not envy any agent in Y . We can conclude that $\phi(X) < \phi(Y)$ or the structure of the champion graph in Y is better than in X . In the latter case, we will find a Pareto improvable cycle in the champion graph of Y , and obtain a new EFX allocation Y' such that $\phi(X) < \phi(Y')$.

1.3 Related Work

Whereas fair division of divisible resources is a classical topic starting from the 1940's [32], fair division of indivisible items has been actively studied in recent years. One of the most popular relaxations of envy-freeness is *envy-freeness up to one item (EF1)* where no agent envies another agent after the removal of *some* item from the other agent's bundle. While the existence of EFX allocations is open, it is known that there always exists an EF1 allocations for any number of agents, and it can be computed in polynomial time [29]. There are a lot of studies on EF1 and EFX [1, 7–9, 13, 14, 16–18, 30, 31]. Another major concept of fairness is *maximin share (MMS)*, which was introduced by Budish [11]. It was shown in [26] that MMS allocations do not always exist, and there have been several studies on approximate MMS allocations [2, 6, 10, 11, 21, 23, 24, 26]. In addition, study on finding *efficient* fair allocations has attracted attention. *Pareto-optimality* is a major notion of efficiency. Caragiannis et al. [14] showed that any allocation that has maximum Nash social welfare is guaranteed to be Pareto-optimal and EF1. Unfortunately, finding an allocation with the maximum Nash social welfare is APX-hard [27]. There are several studies on approximation algorithms for maximizing Nash social welfare [4, 5, 7, 15, 19–22, 28].

There are many real-world scenarios where items or resources need to be divided fairly, e.g., taxi fare division, rent division, task distribution, and so on. Spliddit (www.spliddit.org) is a fair division website, which offers a fair solution for the division of rent, goods, and credit [25]. This website implements mechanisms for users to log in, define what is to be divided, enter their valuations, and demonstrate fair division. Since its launch in 2014, there have been several thousands of users [14]. For more details on Spliddit, we refer to the reader to [25, 31]. Another fair division application is *Course Allocation* used at the Wharton School at the University of Pennsylvania to fairly allocate courses among students [12, 31].

1.4 Organization

In Section 2, we present the model, denote some basic notions introduced by [16, 18], and prove some useful lemmas. In Section 3, we consider EFX with charity for general valuations, and prove Theorem 2. In Section 4, we consider the setting with only one of two types of general valuations, and prove Theorem 1. In Section 5, we construct an instance with $n = 3$ and $m = 7$ that shows a certain limitations of the approach of the lexicographic potential function. The proofs of other results are given in the full version.

2 Preliminaries

Let $N = \{1, \dots, n\}$ be a set of n agents and M be a set of m items. In this paper, we assume that items are indivisible: an item may not be split among multiple agents. Each agent $i \in N$ has a valuation function $v_i : 2^M \rightarrow \mathbb{R}_{\geq 0}$. We assume that (i) any valuation function v_i is *normalized*: $v_i(\emptyset) = 0$ and (ii) it is *monotone*: $S \subseteq T$ implies $v_i(S) \leq v_i(T)$ for any $S, T \subseteq M$.

To simplify notation, we denote $[k]$ by $\{1, \dots, k\}$, write $v_i(g)$ instead of $v_i(\{g\})$ for $g \in M$, and use $S \setminus g, S \cup g$ instead of $S \setminus \{g\}, S \cup \{g\}$, respectively. We also denote $S <_i T$ instead of $v_i(S) < v_i(T)$. In a similar way, we use the symbols $>_i, \leq_i$, and \geq_i .

For $M' \subseteq M$, an *allocation* $X = (X_1, X_2, \dots, X_n)$ on M' is a partition of M' into n disjoint subsets, where X_i is the *bundle* given to agent i . We say that an allocation $X = (X_1, X_2, \dots, X_n)$ on M' is *complete* if $M' = M$. Otherwise, we say that an allocation X is *partial*.

Given an allocation X , we say that agent i *envies* a set of items S if $X_i <_i S$. We say that agent i *envies* agent j if i envies X_j . We say that agent i *EFX envies* a set of items S if there exists some $h \in S$ such that i envies $S \setminus h$. We say that agent i *EFX envies* agent j if i EFX envies X_j . Note that if i EFX envies j then i envies j , but not vice versa. An allocation X is called *envy-free* if no agent envies another. An allocation X is called *EFX* if no agent EFX envies another.

An instance I is a triple $\langle N, M, \mathcal{V} \rangle$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is a set of valuation functions. We use an assumption on instances considered in [16].

► **Definition 4.** An instance I is non-degenerate if for any $i \in N$ and $S, T \subseteq M$,

$$S \neq T \Rightarrow v_i(S) \neq v_i(T).$$

We can show that in order to prove the existence of an EFX allocation, we may assume w.l.o.g. that instances are non-degenerate. This assumption was considered for additive valuations in [16], and we can easily extend it for general valuations. More details are given in the full version. In what follows, *we only deal with non-degenerate instances*.

2.1 Overall Approach

All of our results on the existence of EFX can be viewed in a unified framework as follows: we first fix an appropriate potential function ϕ on all allocations. We then show that given any partial EFX allocation X , one can construct a new EFX allocation Y that makes progress with respect to the potential function, i.e., $\phi(X) < \phi(Y)$. Since there are finitely many allocations, there must exist a complete EFX allocation. One of natural potential functions is *social welfare*. Given an allocation X , we denote the *social welfare* of X by $\varphi(X) = \sum_{i \in N} v_i(X_i)$. A typical notion to progress the social welfare is Pareto domination. Given two allocations X, Y , we say that Y *Pareto dominates* X if $Y_i \geq_i X_i$ for all $i \in N$, and $Y_j >_j X_j$ for some agent $j \in N$. Clearly, if Y Pareto dominates X , then $\varphi(Y) > \varphi(X)$. Chaudhury et al. [16] have shown that there does not always exist a Pareto-dominating EFX allocation when $n = 3$ for additive case. To overcome this barrier they introduce a *lexicographic potential function*, which we also use to prove Theorems 2 and 3. In addition, to prove Theorem 1, we use a new potential function. More detail on each potential function is presented in Sections 3 and 4.

2.2 Minimum Preferred Set and Most Envious Agent

Most envious agent is a basic notion, introduced in [18]. Consider an allocation $X = (X_1, X_2, \dots, X_n)$ and a set $S \subseteq M$. For an agent i such that $S >_i X_i$, we define a *minimum preferred set* $P_X(i, S)$ of agent i for S with respect to allocation X as a smallest cardinality subset S' of S such that $S' >_i X_i$. Define $\kappa_X(i, S)$ by

$$\kappa_X(i, S) = \begin{cases} |P_X(i, S)| & \text{if } S >_i X_i, \\ +\infty & \text{otherwise.} \end{cases}$$

Let $\kappa_X(S) = \min_{i \in N} \kappa_X(i, S)$. We define $A_X(S)$ for a set S as the set of agents with the smallest values of $\kappa_X(i, S)$, i.e.,

$$A_X(S) = \{i \in N \mid S >_i X_i \text{ and } \kappa_X(i, S) = \kappa_X(S)\}.$$

We call $A_X(S)$ the set of *most envious agents*. Note that if agent i is a most envious agent for S , it holds that i envies $P_X(i, S)$ and no agent EFX envies $P_X(i, S)$. When it is clear from the context, we abbreviate $P_X(i, S)$ as P .

2.3 Champions and Champion Graph

In order to find a new EFX allocation from the existing partial EFX allocation, Champions and Champion graph are important notions, introduced in [16]. Let X be a partial allocation on $M' \subsetneq M$ and let $g \in M \setminus M'$ be an unallocated item. For agents i and j (possibly $i = j$), we say that i *g-champions* j if i is a most envious agent for $X_j \cup g$. Then, we also call i a *g-champion* of j . When i is a most envious agent for $X_i \cup g$, we call i a *self g-champion*. Note that every agent j has a *g-champion*. Indeed, since instances are non-degenerate, and valuations are monotone, we have $X_j \cup g >_j X_j$. That is, since j envies $X_j \cup g$, there exists at least one most envious agent for $X_j \cup g$.

We say that i *g-decomposes* j if i *g-champions* j , and $\{g\} \subsetneq P \subsetneq X_j \cup g$, where P is a minimum preferred set of i for $X_j \cup g$. When i *g-decomposes* j , we can decompose X_j into $P \setminus g$ and $(X_j \cup g) \setminus P$. If there is no ambiguity, then $T_j = P \setminus g$ and $B_j = (X_j \cup g) \setminus P$ are called *top and bottom half-bundles* of X_j , respectively. The following lemma illustrates a typical situation such that i *g-decomposes* j .

► **Lemma 5.** *If i g-champions j , i does not envy j , and both i and j are not self g-champions, then i g-decomposes j .*

Proof. By the assumption, we have $X_j <_i X_i <_i X_j \cup g$. Let P be a minimum preferred set of i for $X_j \cup g$. If $g \notin P$, then $P \subseteq X_j$, and by the monotonicity, we have $P \leq_i X_j <_i X_i$. This contradicts the definition of P . Thus, $g \in P$. If $P = \{g\}$, then $\kappa_X(i, X_i \cup g) = 1$, and hence it contradicts that i is not a self *g-champion*. Thus, $\{g\} \subsetneq P$. Furthermore if $P = X_j \cup g$, then $\kappa_X(X_j \cup g) = |P|$ and hence it contradicts that j is not a self *g-champion*. Therefore $\{g\} \subsetneq P \subsetneq X_j \cup g$, and thus i *g-decomposes* j . ◀

► **Definition 6.** *The champion graph $M_X = (N, E)$ with respect to allocation X is a labeled directed multi-graph. The vertices correspond to the agents, and E consists of the following two types of edges:*

1. *Envy edges:* $i \rightarrow j$ iff i envies j .
2. *Champion edges:* $i \xrightarrow{g} j$ iff i *g-champions* j , where g is an unallocated item.

Envy graph which consists of only envy edges is introduced in [29]. The original champion graph considered in [16] consists of only champion edges. Our definition of champion graph combines these two notions for convenience. Recently, Berger et al. [8] denote the generalized champion graph that contains more additional edges. For convenience, $i \rightarrow j$ and $i \xrightarrow{g} j$ are sometimes denoted by $i \xrightarrow{\emptyset} j$ and $i \xrightarrow{\{g\}} j$, respectively. Berger et al. [8] denote the notion of Pareto improvable cycle, which is very useful in our argument.

► **Definition 7.** A cycle $C = a_1 \xrightarrow{H_1} a_2 \xrightarrow{H_2} \dots \xrightarrow{H_{k-1}} a_k \xrightarrow{H_k} a_1$ in M_X is called Pareto improvable (PI) if for every $i, j \in [k]$ we have $H_i \cap H_j = \emptyset$, where H_i is an empty set or a singleton of an unallocated item.

The following lemma shows that if we have a Pareto improvable cycle in M_X , then there exists an allocation Y that Pareto dominates X while keeping EFX.

► **Lemma 8** (Berger et al. [8]). Let X be an allocation. If M_X contains a Pareto improvable cycle, then there exists an allocation Y Pareto dominating X such that for any $i, j \in N$, if i does not EFX envy j in X , then neither in Y . In particular, if X is an EFX allocation, then so is Y . Furthermore, every agent i along the cycle satisfies $X_i <_i Y_i$.

► **Corollary 9.** Let X be an EFX allocation. If M_X contains an envy-cycle³, a self g -champion, or a cycle composed of envy edges and at most one champion edge, then there exists an EFX allocation Y that Pareto dominates X .

3 Existence of EFX with at most $n - 2$ unallocated items

In this section, we prove Theorem 2. We use a lexicographic potential function as in [16]. Recall that $N = \{1, \dots, n\}$. For an allocation X , the *lexicographic potential function* $\phi(X)$ is defined as the vector $(v_1(X_1), \dots, v_n(X_n))$. Intuitively, agent 1 is the most important agent and agent n is the least important agent in N .

► **Definition 10.** For two allocations X, Y , We denote $Y \succ_{\text{lex}} X$ if $\phi(Y)$ is lexicographically larger than $\phi(X)$, i.e., for some $k \in N$, we have that $Y_j =_j X_j$ for all $1 \leq j < k$, and $Y_k >_k X_k$.

Note that if Y Pareto dominates X then $Y \succ_{\text{lex}} X$, but not vice versa. The following basic lemma is shown in [8], which we also use.

► **Lemma 11** (Berger et al. [8]). If for every partial EFX allocation X with k unallocated items, there exists an EFX allocation Y such that $Y \succ_{\text{lex}} X$, then there exists an EFX allocation with at most $k - 1$ unallocated items. Moreover, no agent envies the set of $k - 1$ unallocated items.

By Lemma 11, in order to prove Theorem 2, it suffices to show that for every partial EFX allocation X with at least $n - 1$ unallocated items, there exists an EFX allocation Y such that $Y \succ_{\text{lex}} X$. We first prove the following lemma, which is used in the proof of Theorem 2.

³ envy-cycle is a dicycle in M_X composed of only envy edges.

► **Lemma 12.** *Let X be an EFX allocation with at least $n - 1$ unallocated items. Then, there exists an EFX allocation Y Pareto dominating X in the following two cases.*

- (1) *the number of unallocated items is at least n .*
- (2) *there exists at least one envy edge $j \rightarrow i$ in M_X .*

Moreover, in case (2), some agent $l \in N \setminus i$ is strictly better off than in X , i.e., $Y_l >_l X_l$.

Proof. Let $\{g_1, \dots, g_k\}$ denote the set of unallocated items. We first prove the case of (1). It suffices to prove that there exists a PI cycle in M_X by Lemma 8. Let a_1 be an arbitrary agent. Then, some agent a_2 g_1 -champions a_1 . If $a_2 = a_1$, then we have a PI cycle, and we are done. Assume that $a_2 \neq a_1$. Then, some agent a_3 g_2 -champions a_2 . If $a_3 = a_1$ or a_2 , then we have a PI cycle. Indeed, in the first case we have a cycle $a_1 \xrightarrow{g_2} a_2 \xrightarrow{g_1} a_1$, and in the second case we have a self g_2 -champion. We can continue this way to conclude that w.l.o.g. we have a directed path $a_n \xrightarrow{g_{n-1}} a_{n-1} \xrightarrow{g_{n-2}} \dots \xrightarrow{g_1} a_1$ in M_X , where a_1, \dots, a_n are different agents. Now, some agent g_n -champions a_n in M_X . No matter who it is, there exists a PI cycle, and we are done.

We prove the case of (2) in a similar way. Assume w.l.o.g. that some agent a_2 envies a_1 . By a similar argument as above, we can conclude that w.l.o.g. we have a directed path $a_n \xrightarrow{g_{n-1}} a_{n-1} \xrightarrow{g_{n-2}} \dots \xrightarrow{g_3} a_3 \xrightarrow{g_2} a_2 \rightarrow a_1$ in M_X , where a_1, \dots, a_n are different agents. Now, some agent g_1 -champions a_n . No matter who it is, there exists a PI cycle, and we are done. Moreover, in any cases, we have a PI cycle containing some agent in $N \setminus a_1$. Hence, the last statement of lemma holds by Lemma 8. ◀

We are now ready to prove Theorem 2. We fix an arbitrary ordering of the agents.

Proof of Theorem 2. Let X be an EFX allocation with $k \geq n - 1$ unallocated items, and let $\{g_1, \dots, g_k\}$ denote the set of unallocated items. By Lemma 11, it suffices to prove that there exists an EFX allocation Y such that $Y \succ_{\text{lex}} X$. By Lemma 12, when $k \geq n$, or $k = n - 1$ and there exists at least one envy edge in M_X , we are done. Assume that $k = n - 1$ and there exists no envy edge in M_X . Let a_1 be the last agent in the fixed ordering, i.e., a_1 is the least important agent in the lexicographic potential function. By a similar argument in Lemma 12, we conclude that w.l.o.g. we have a directed path $a_n \xrightarrow{g_{n-1}} a_{n-1} \xrightarrow{g_{n-2}} \dots \xrightarrow{g_1} a_1$ in M_X , where a_1, \dots, a_n are different agents. Furthermore, we may assume that there are no self g_i -champions for $1 \leq i \leq n - 1$ since otherwise we have a PI-cycle. Since there are no self-champions, and there exists no envy edge in M_X , a_{i+1} g_i -decomposes a_i for $1 \leq i \leq n - 1$ by Lemma 5. Let T_i and B_i are the top and bottom half-bundles of X_{a_i} decomposed by a_{i+1} for $1 \leq i \leq n - 1$, respectively. Consider $Z = \max_{a_1} \{T_1 \cup g_1, T_2 \cup g_2, \dots, T_{n-1} \cup g_{n-1}, X_{a_2}, \dots, X_{a_n}\}^4$. We discuss in two cases.

Case 1: $Z = T_i \cup g_i$ or X_{a_i} for $2 \leq i \leq n$

We define a new allocation X' as follows:

$$\begin{aligned} X'_{a_1} &= Z, \\ X'_{a_j} &= T_{j-1} \cup g_{j-1} && \text{for } 2 \leq j \leq i, \\ X'_{a_j} &= X_{a_j} && \text{for } i < j \leq n. \end{aligned}$$

We show that X' is EFX and $X' \succ_{\text{lex}} X$. For $1 \leq t \leq i - 1$, since $T_t \cup g_t$ is a minimum preferred set of a_{t+1} and a_{t+1} is a most envious agent for $T_t \cup g_t$, no agent EFX envies

⁴ $\max_{a_1} \{T_1 \cup g_1, T_2 \cup g_2, \dots, T_{n-1} \cup g_{n-1}, X_{a_2}, \dots, X_{a_n}\}$ is a_1 's most favorite bundle out of $T_1 \cup g_1, T_2 \cup g_2, \dots, T_{n-1} \cup g_{n-1}, X_{a_2}, \dots, X_{a_n}$.

$T_t \cup g_t$ in X . Thus, for $2 \leq s \leq n$, since $X'_{a_s} \geq_{a_s} X_{a_s}$, agent a_s does not EFX envy $T_t \cup g_t$ in X' . For $2 \leq s \leq n$, since X is envy-free and the fact that $X'_{a_s} \geq_{a_s} X_{a_s}$, agent a_s does not envy X_{a_u} for $1 \leq u \leq n$ in X' . By the definition of Z , a_1 does not envy any agents in X' . Therefore, X' is EFX. Furthermore, for $2 \leq j \leq i$, each agent a_j is strictly better off than in X , and each agent a_j does not change her bundle for $i < j \leq n$. Thus, we have $X' \succ_{\text{lex}} X$, and we are done.

Case 2: $Z = T_1 \cup g_1$

We define a new allocation X' as follows:

$$\begin{aligned} X'_{a_1} &= Z, \\ X'_{a_i} &= X_{a_i} \quad \text{for } 2 \leq i \leq n. \end{aligned}$$

We show that X' is EFX. Since we change only a_1 's bundle from X , it is enough to check that there is no EFX envy from or to a_1 . By the definition of Z , a_1 does not envy any agent in X' . Since $Z = T_1 \cup g_1$ is a minimum preferred set of a_2 for $X_{a_1} \cup g_1$, and a_2 is a most envious agent for $X_{a_1} \cup g_1$, no agent EFX envies $T_1 \cup g_1$ in X' . Thus X' is EFX. In addition, since $Z = T_1 \cup g_1$ is a minimum preferred set of a_2 , a_2 envies a_1 in X' . By the fact that $B_1 \neq \emptyset$, we now have at least $n - 1$ items in $\{g_2, \dots, g_{n-1}\} \cup B_1$ that are unallocated. Thus by the case of (2) in Lemma 12, there exists an EFX allocation X'' that Pareto dominates X' . Furthermore, there exists some agent a_i ($2 \leq i \leq n$) such that $X''_{a_i} >_{a_i} X'_{a_i} = X_{a_i}$. Since $X''_{a_j} \geq_{a_j} X'_{a_j} = X_{a_j}$ for $2 \leq j \leq n$, we have $X'' \succ_{\text{lex}} X$, and we are done. \blacktriangleleft

4 Existence of EFX with One of Two General Valuations

In this section, we prove Theorem 1. For two general valuation functions v_a and v_b , let N_a (resp. N_b) be the set of agents whose valuation is v_a (resp. v_b). To prove Theorem 1, we introduce a new potential function. For an allocation X , we write $N_a = \{a_0, a_1, \dots, a_s\}$ and $N_b = \{b_0, b_1, \dots, b_t\}$, where $X_{a_0} <_a X_{a_1} <_a \dots <_a X_{a_s}$ and $X_{b_0} <_b X_{b_1} <_b \dots <_b X_{b_t}$. Define the *partition leximin potential function* $\psi(X)$ as the vector $(v_a(X_{a_0}), \dots, v_a(X_{a_s}), v_b(X_{b_0}), \dots, v_b(X_{b_t}))$.

► **Definition 13.** For two allocations X, Y , we denote $Y \succ_{\text{p.lexmin}} X$ if $\psi(Y)$ is lexicographically larger than $\psi(X)$.

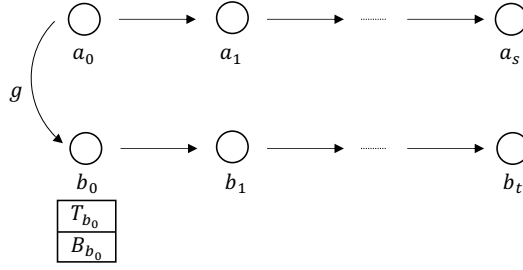
That is, we prioritize N_a over N_b , compare agents in N_a by the leximin ordering, and second compare agents in N_b by the leximin ordering. Note that if Y Pareto dominates X , then $Y \succ_{\text{p.lexmin}} X$ but not vice versa. Our goal is to show the following theorem.

► **Theorem 14.** Let X be a partial EFX allocation. Then, there exists an EFX allocation Y such that $Y \succ_{\text{p.lexmin}} X$.

If Theorem 14 holds, then since there are finitely many allocations, there must exist a complete EFX allocation, and thus Theorem 1 holds.

We say that an allocation X is *semi-EFX* if there can be EFX envy only among agents belonging to N_b in X , i.e., no agent belonging to N_a EFX envies any agents, and no agent belonging to N_b EFX envies any agent belonging to N_a in X . The following lemma shows that if we have a semi-EFX allocation, then we can obtain an EFX allocation such that all the agents in N_a and $b_0 \in N_b$ do not change their bundles.

► **Lemma 15.** Let X be a semi-EFX allocation such that $X_{b_0} <_b X_{b_1} <_b \dots <_b X_{b_t}$. Then, there exists an EFX allocation Y such that $Y_{a_i} = X_{a_i}$ for any agent $a_i \in N_a$, and $Y_{b_0} = X_{b_0}$.



■ **Figure 1** The champion graph M_X (the edges are only partially drawn) in the proof of Theorem 14. Every agent other than a_0 and b_0 is envied by some other agents, and a_0 g -decomposes b_0 in M_X .

Proof. If X is EFX, then the lemma obviously holds. Assume that X is not EFX. Then, for some two agents $b_i, b_j \in N_b$, b_i EFX envies b_j in X . Thus, there exists an item $h \in X_{b_j}$ such that $X_{b_i} <_b X_{b_j} \setminus h$. We define a new allocation X' as $X'_{b_j} = X_{b_j} \setminus h$, and $X'_k = X_k$ for any $k \in N \setminus b_j$. Then, X' is also semi-EFX. Indeed, since we only change b_j 's bundle, it suffice to consider EFX envy from or to b_j . Since b_j 's bundle is a subset of X_{b_j} , and valuations are monotone, agents who do not EFX envy b_j in X do not EFX envy b_j either in X' . In addition, since $X_{b_i} = X'_{b_i} <_b X'_{b_j}$, and b_i does not EFX envy any agent belonging to N_a in X , b_j does not EFX envy any agent in N_a either in X' . Therefore X' is also semi-EFX. Furthermore since b_j is not b_0 , we have $X'_{b_0} = X_{b_0}$. If X' is EFX, then we are done. Otherwise, since the number of all items allocated is decreasing, we can continue this way to obtain an EFX allocation Y such that $Y_{a_i} = X_{a_i}$ for any agent $a_i \in N_a$, and $Y_{b_0} = X_{b_0}$. ◀

We are now ready to prove Theorem 14.

Proof of Theorem 14. Let X be a partial EFX allocation and let g be an unallocated item. Define $N_a = \{a_0, a_1, \dots, a_s\}$ and $N_b = \{b_0, b_1, \dots, b_t\}$, where $X_{a_0} <_a X_{a_1} <_a \dots <_a X_{a_s}$ and $X_{b_0} <_b X_{b_1} <_b \dots <_b X_{b_t}$. If there exists a PI cycle in M_X , then we are done by Lemma 8. Assume that there is no PI cycle in M_X . We first show that a_0 g -decomposes b_0 . By the assumption, neither a_0 nor b_0 is a self g -champion. If a_0 envies b_0 , then every agent other than a_0 is envied by some other agents. Some agent g -champions a_0 . No matter who it is, there exists a PI cycle in M_X , and this is a contradiction. Thus, a_0 does not envy b_0 . Now, some agent i g -champions b_0 . If $i \in N_b$, then since $X_{b_0} \leq_b X_i$, we have $\kappa_X(b_0, X_{b_0} \cup g) \leq \kappa_X(i, X_{b_0} \cup g) = \kappa_X(X_{b_0} \cup g)$. This implies that b_0 is a self g -champion, and this is a contradiction. Hence, we have $i \in N_a$. Then, since $X_{a_0} \leq_a X_i$, we have $\kappa_X(a_0, X_{b_0} \cup g) \leq \kappa_X(i, X_{b_0} \cup g) = \kappa_X(X_{b_0} \cup g)$. Hence, a_0 g -champions b_0 . As a result, a_0 g -decomposes b_0 by Lemma 5. Therefore, X_{b_0} is decomposed into top and bottom half-bundles. Let T_{b_0} and B_{b_0} be the top and bottom half-bundles of X_{b_0} . Figure 1 partially illustrates M_X . For $0 \leq i \leq s$, we define $U_{a_i} \subseteq X_{a_i}$ as follows:

$$U_{a_i} = \begin{cases} X_{a_i} & \text{if } b_0 \text{ does not envy } a_i \text{ in } X, \\ \hat{X}_{a_i} & \text{otherwise,} \end{cases}$$

where, \hat{X}_{a_i} is a maximum cardinality proper subset of X_{a_i} maximizing $v_b(\hat{X}_{a_i})$. Note that we have $|\hat{X}_{a_i}| = |X_{a_i}| - 1$. Consider $Z = \max_b \{T_{b_0} \cup g, U_{a_0}, \dots, U_{a_s}\}$. We define a new allocation X' as follows:

$$\begin{aligned}
X'_{a_i} &= \begin{cases} T_{b_0} \cup g & \text{if } Z = U_{a_i} \\ X_{a_i} & \text{otherwise} \end{cases} & \text{for } 0 \leq i \leq s, \\
X'_{b_0} &= Z, \\
X'_{b_j} &= X_{b_j} & \text{for } 1 \leq j \leq t.
\end{aligned}$$

We can easily check that X' is a legal allocation. That is, $X'_{a_0}, \dots, X'_{a_s}, X'_{b_0}, \dots, X'_{b_t}$ is a partition of a subset of M . We show that X' is semi-EFX.

- *Any two agents in N_a do not EFX envy each other:* Note that since $X_{a_0} <_a T_{b_0} \cup g$ and $X_{a_0} <_a X_{a_k}$ for $1 \leq k \leq s$, we have $X_{a_0} \leq_a X'_{a_i}$ for $0 \leq i \leq s$. Let a_i and $a_{i'}$ be two agents in N_a . If $X'_{a_{i'}} = X_{a_{i'}}$, then since $X_{a_0} \leq_a X'_{a_i}$ and by the fact that X is EFX, a_i does not EFX envy $a_{i'}$ in X' . If $X'_{a_{i'}} = T_{b_0} \cup g$, then since $X_{a_0} \leq_a X'_{a_i}$ and a_0 does not EFX envy $T_{b_0} \cup g$ in X , a_i does not EFX envy $a_{i'}$ in X' .
- *Any agent in N_a does not EFX envy any agent in $N_b \setminus b_0$:* Since $X_{a_0} \leq_a X'_{a_i}$ for $0 \leq i \leq s$, X is EFX, and any agent in $N_b \setminus b_0$ does not change her bundle, any agent in N_a does not EFX envy any agent in $N_b \setminus b_0$.
- *Any agent in N_a does not EFX envy b_0 :* If $Z = T_{b_0} \cup g$, then since $X_{a_0} \leq_a X'_{a_i}$ for $0 \leq i \leq s$ and a_0 does not EFX envy $T_{b_0} \cup g$ in X , any agent in N_a does not EFX envy b_0 in X' . If $Z = U_{a_k}$ for some $0 \leq k \leq s$, then since $X_{a_0} \leq_a X'_{a_i}$ for $0 \leq i \leq s$, and a_0 does not EFX envy $U_{a_k} \subseteq X_{a_k}$, any agent in N_a does not EFX envy b_0 in X' .
- *Any agent in $N_b \setminus b_0$ does not EFX envy any agent in N_a :* Let a_i be any agent in N_a and let b_j be any agent in $N_b \setminus b_0$. If $X'_{a_i} = X_{a_i}$, then since $X'_{b_j} = X_{b_j}$ and X is EFX, b_j does not EFX envy a_i . If $X'_{a_i} = T_{b_0} \cup g$, then since b_0 is not a self g -champion in X , we have $T_{b_0} \cup g <_b X_{b_j}$. Thus b_j does not envy a_i in X' .
- *b_0 does not EFX envy any agent in N_a :* Let a_i be any agent in N_a . If $X'_{a_i} = T_{b_0} \cup g$, then since $Z = \max_b \{T_{b_0} \cup g, U_{a_0}, \dots, U_{a_s}\} \geq_b T_{b_0} \cup g$, b_0 does not envy a_i in X' . If $X'_{a_i} = X_{a_i}$, then since $Z = \max_b \{T_{b_0} \cup g, U_{a_0}, \dots, U_{a_s}\} \geq_b U_{a_i}$, for any proper subset S of X_{a_i} , we have $Z \geq_b U_{a_i} \geq_b S$ by the definition of U_{a_i} and \hat{X}_{a_i} . Thus b_0 does not EFX envy a_i in X' .

Therefore X' is semi-EFX. By Lemma 15, there exists an EFX allocation X'' such that $X''_{a_i} = X'_{a_i}$ for $0 \leq i \leq s$, and $X''_{b_0} = X'_{b_0}$. We discuss in the following three cases.

Case 1: $Z = U_{a_0}$

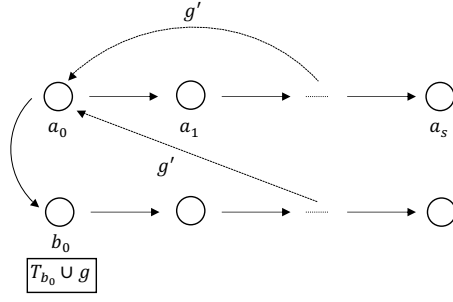
In this case, we have $X''_{a_0} = X'_{a_0} = T_{b_0} \cup g >_a X_{a_0}$ and $X''_{a_k} = X'_{a_k} = X_{a_k}$ for $1 \leq k \leq s$. Thus, we have $X'' \succ_{p.\text{lexmin}} X$, and we are done.

Case 2: $Z = T_{b_0} \cup g$

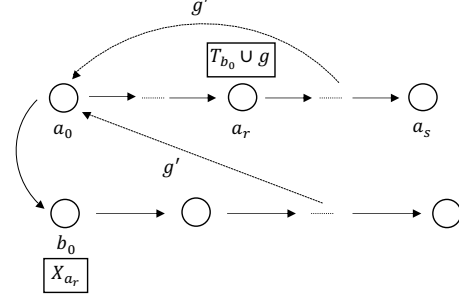
In this case, since we have $X''_{a_0} = X'_{a_0} = X_{a_0} <_a T_{b_0} \cup g = X'_{b_0} = X''_{b_0}$, a_0 envies b_0 in X'' . Thus, every agent other than a_0 is envied by some other agents in X'' . By the fact that $B_{b_0} \neq \emptyset$, there is an unallocated item $g' \in B_{b_0}$. Then, some agent l g' -champions a_0 (see Figure 2). If $l = a_i \in N_a$, then by following agents in N_a backwards we obtain a PI cycle $a_0 \rightarrow \dots \rightarrow a_{i-1} \rightarrow a_i \xrightarrow{g'} a_0$ in $M_{X''}$. If $l \in N_b$, then by following agents in N_b backwards we also obtain a PI cycle $a_0 \rightarrow b_0 \rightarrow \dots \rightarrow l \xrightarrow{g'} a_0$ in $M_{X''}$. Therefore, in either case, there exists a PI cycle containing a_0 in $M_{X''}$. By Lemma 8, there exists an EFX allocation X''' such that $X'''_{a_0} >_a X''_{a_0} = X'_{a_0} = X_{a_0}$ and $X'''_{a_k} \geq_a X''_{a_k} = X'_{a_k} = X_{a_k}$ for $1 \leq k \leq s$. Therefore, we have $X''' \succ_{p.\text{lexmin}} X$, and we are done.

Case 3: $Z = U_{a_r}$ for some $1 \leq r \leq s$

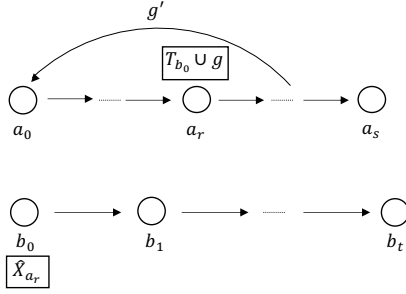
In this case, if $U_{a_r} = X_{a_r}$, then since we have $X''_{a_0} = X'_{a_0} = X_{a_0} <_a X_{a_r} = X'_{b_0} = X''_{b_0}$, a_0 envies b_0 in X'' . By the fact that $B_{b_0} \neq \emptyset$, there is an unallocated item $g' \in B_{b_0}$. In a similar way to Case 2, the fact that some agent g' -champions a_0 implies that there exists a PI cycle containing a_0 in $M_{X''}$ (see Figure 3). By Lemma 8, there



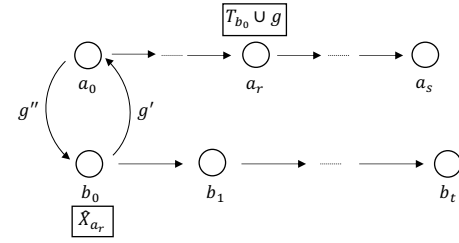
■ **Figure 2** The champion graph $M_{X''}$ (the edges are only partially drawn) in Case 2. Every agent other than a_0 is envied by some other agents, and some agent g' -champions a_0 in $M_{X''}$.



■ **Figure 3** The champion graph $M_{X''}$ (the edges are only partially drawn) in the case where $U_{a_r} = X_{a_r}$ in Case 3. Every agent other than a_0 is envied by some other agents, and some agent g' -champions a_0 in $M_{X''}$.



■ **Figure 4** The champion graph $M_{X'}$ (the edges are only partially drawn) in the case where $U_{a_r} = \hat{X}_{a_r}$, and some agent in N_a g' -champions a_0 in Case 3.



■ **Figure 5** The champion graph $M_{X'}$ (the edges are only partially drawn) in the case where $U_{a_r} = \hat{X}_{a_r}$, and some agent in N_b g' -champions a_0 in Case 3. b_0 g' -champions a_0 , and a_0 g'' -champions b_0 in $M_{X'}$.

exists an EFX allocation X''' such that $X'''_{a_0} >_a X''_{a_0}$ and $X'''_{a_k} \geq_a X''_{a_k}$ for $1 \leq k \leq s$. Thus, we have $X'''_{a_k} \geq_a X''_{a_k} = X'_{a_k} = X_{a_k} >_a X_{a_0}$ for $1 \leq k \leq s$ with $k \neq r$, and $X'''_{a_r} \geq_a X''_{a_r} = X'_{a_r} = T_{b_0} \cup g >_a X_{a_0}$. That is, $X'''_{a_i} >_a X_{a_0}$ for $0 \leq i \leq s$. Therefore, we have $X''' \succ_{\text{p.lexmin}} X$, and we are done.

If $U_{a_r} = \hat{X}_{a_r}$, then we consider semi-EFX allocation X' , not X'' in this case. some agent l g' -champions a_0 in $M_{X'}$. If $l \in N_a$, then by following agents in N_a backwards we obtain a PI cycle $a_0 \rightarrow \dots \rightarrow l \xrightarrow{g'} a_0$ in $M_{X''}$ (see Figure 4). By Lemma 8, there exists a semi-EFX allocation X''' such that $X'''_{a_0} >_a X'_{a_0}$ and $X'''_{a_k} \geq_a X'_{a_k}$ for $1 \leq k \leq s$. By a similar argument as above, we have $X'''_{a_i} >_a X_{a_0}$ for $0 \leq i \leq s$. By Lemma 15, there exists an EFX allocation X'''' such that $X''''_{a_i} = X'''_{a_i}$ for $0 \leq i \leq s$. Therefore, we have $X''''_{a_i} >_a X_{a_0}$ for $0 \leq i \leq s$, thus we have $X'''' \succ_{\text{p.lexmin}} X$, and we are done.

If $l = b_j \in N_b$, then since $X'_{b_0} \leq_b X'_{b_j}$, we have $\kappa_{X'}(b_0, X'_{a_0} \cup g') \leq \kappa_{X'}(b_j, X'_{a_0} \cup g') = \kappa_{X'}(X'_{a_0} \cup g')$. Thus b_0 g' -champions a_0 in X' . Since $U_{a_r} = \hat{X}_{a_r}$, there exists an unallocated item $g'' \in X_{a_r} \setminus \hat{X}_{a_r}$. Note that we have $X_{a_r} = X'_{b_0} \cup g''$. We claim that a_0 g'' -champions b_0 in X' . Indeed, since any agent $u \in N \setminus \{a_r, b_0\}$ does not change her bundle, and X is EFX, u does not EFX envy $X_{a_r} = X'_{b_0} \cup g''$ in X' . In addition, since $X'_{a_r} = T_{b_0} \cup g >_a X_{a_0}$, and a_0 does not EFX envy X_{a_r} in X , a_r does not EFX envy X_{a_r} in X' . Furthermore, since $X'_{b_0} = U_{a_r} = \hat{X}_{a_r}$ is a maximum cardinality proper subset of X_{a_r} maximizing $v_b(\hat{X}_{a_r})$, b_0 does not EFX envy X_{a_r} in X' . To sum up, for

any proper subset S of $X'_{b_0} \cup g''$, any agent in N does not envy S in X' . Furthermore, since we have $X'_{a_0} = X_{a_0} <_a X_{a_r} = X'_{b_0} \cup g''$, a_0 envies $X'_{b_0} \cup g''$ in X' . Thus, since $\kappa_{X'}(a_0, X'_{b_0} \cup g'') = |X'_{b_0} \cup g''| \leq \kappa_{X'}(w, X'_{b_0} \cup g'')$ for any $w \in N$, a_0 is a most envious agent for $X'_{b_0} \cup g''$. That is, a_0 g'' -champions b_0 in X' (see Figure 5). We now have a PI cycle $a_0 \xrightarrow{g''} b_0 \xrightarrow{g'} a_0$ in $M_{X'}$, and by Lemma 8, we obtain a semi-EFX allocation X''' such that $X'''_{a_0} >_a X'_{a_0}$ and $X'''_{a_k} = X'_{a_k}$ for $1 \leq k \leq s$. By a similar argument as above, we have $X'''_{a_i} >_a X_{a_0}$ for $0 \leq i \leq s$. By Lemma 15, there exists an EFX allocation X'''' such that $X''''_{a_i} = X'''_{a_i}$ for $0 \leq i \leq s$. Therefore, we have $X''''_{a_i} >_a X_{a_0}$ for $0 \leq i \leq s$, thus we have $X'''' \succ_{\text{p.lexmin}} X$, and we are done. \blacktriangleleft

5 Limitations of the Lexicographic Potential Function

In Section 3, in order to prove the existence of EFX, we show that given a partial EFX allocation X , there exists an EFX allocation Y such that $Y \succ_{\text{lex}} X$. Recently, Chaudhury et al. [17] have shown that there does not always exist a lexicographically larger EFX allocation when $n = 4$ for additive valuations. In this section, we show that there does not always exist a lexicographically larger EFX allocation when $n = 3$ for general valuations. Thus, the approach using the lexicographic potential function is not sufficient to show the existence of EFX even when $n = 3$ for general valuations.

The following theorem shows that there exist an instance and a partial EFX allocation X such that no complete EFX allocation Y such that $Y \succ_{\text{lex}} X$.

► Theorem 16. *There exist an instance I with three agents, $\{1, 2, 3\}$ with general valuations, seven items $\{g_i \mid i \in [7]\}$, and a partial EFX allocation X , such that in every complete EFX allocation, the valuation of agent 1 will be strictly worse off than in X .*

Proof. We partially define the conditions of each agent's valuation function. Assume that agent 1 has an additive valuation satisfying the following conditions: $v_1(g_1) = v_1(g_2) > 0, v_1(g_i) = 0$ for $3 \leq i \leq 7$. Agent 2 has a general valuation satisfying following four conditions:

- (1) $\{g_i\} <_2 \{g_1\}$ for $2 \leq i \leq 7$
- (2) $\{g_i, g_j\} <_2 \{g_1\}$ for $2 \leq i < j \leq 7$ and $(i, j) \neq (3, 4), (5, 7)$
- (3) $\{g_4, g_5, g_6\} <_2 \{g_1\} <_2 \{g_3, g_4\} <_2 \{g_5, g_7\}$
- (4) $\{g_5, g_7\} <_2 \{g_1, g_i\}$ for $2 \leq i \leq 7$

Similarly, agent 3 has a general valuation satisfying the following four conditions:

- (1') $\{g_i\} <_3 \{g_1\}$ for $2 \leq i \leq 7$
- (2') $\{g_i, g_j\} <_3 \{g_1\}$ for $2 \leq i < j \leq 7$ and $(i, j) \neq (5, 6), (3, 7)$
- (3') $\{g_3, g_4, g_6\} <_3 \{g_1\} <_3 \{g_5, g_6\} <_3 \{g_3, g_7\}$
- (4') $\{g_3, g_7\} <_3 \{g_1, g_i\}$ for $2 \leq i \leq 7$

Note that all conditions do not violate the monotonicity of valuation functions. We now consider a partial allocation $X = (\{g_1, g_2\}, \{g_3, g_4\}, \{g_5, g_6\})$. We can easily check that X is an EFX allocation. Consider any complete EFX allocation Y . We show that $Y_1 <_1 X_1$. Assume that $X_1 \leq_1 Y_1$. Then, it must be $\{g_1, g_2\} \subseteq Y_1$ by the definition of 1's valuation. If $\{g_1, g_2\} \subsetneq Y_1$, then at least one of agents 2 and 3 has a bundle of size at most 2. If $|Y_2| \leq 2$, then since $Y_2 <_2 \{g_1, g_2\}$ by (2), (3), and (4), agent 2 EFX envies agent 1. This is a contradiction. The similar argument holds when $|Y_3| \leq 2$. Thus, we have $Y_1 = \{g_1, g_2\}$. Therefore, by (1) and (1'), both agent 2 and agent 3 have bundles of size at least 2. This implies that $|Y_2| = 2$ and $|Y_3| = 3$, or $|Y_2| = 3$ and $|Y_3| = 2$.

If $|Y_2| = 2$ and $|Y_3| = 3$, then since agent 2 does not EFX envy agent 1, $Y_2 = \{g_3, g_4\}$ or $\{g_5, g_7\}$ by (2). If $Y_2 = \{g_3, g_4\}$ then $Y_3 = \{g_5, g_6, g_7\}$, and agent 2 EFX envies agent 3 by (3). This is a contradiction. If $Y_2 = \{g_5, g_7\}$ then $Y_3 = \{g_3, g_4, g_6\}$, and agent 3 EFX envies agent 1 by (3'). This is a contradiction. The similar argument holds when $|Y_2| = 3$ and $|Y_3| = 2$. As a result, we conclude that $Y_1 <_1 X_1$, and thus the value of agent 1 will be strictly worse off than in X . ◀

References

- 1 Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, Alexandros Hollender, and Alexandros A Voudouris. Maximum Nash welfare and other stories about efx. *Theoretical Computer Science*, 863:69–85, 2021.
- 2 Georgios Amanatidis, Evangelos Markakis, Afshin Nikzad, and Amin Saberi. Approximation algorithms for computing maximin share allocations. *ACM Transactions on Algorithms (TALG)*, 13(4):1–28, 2017.
- 3 Georgios Amanatidis, Evangelos Markakis, and Apostolos Ntokos. Multiple birds with one stone: Beating $1/2$ for EFX and GMMS via envy cycle elimination. *Theoretical Computer Science*, 841:94–109, 2020.
- 4 Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V Vazirani. Nash social welfare for indivisible items under separable, piecewise-linear concave utilities. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2274–2290, 2018.
- 5 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash social welfare, matrix permanent, and stable polynomials. In *8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 1–12, 2017.
- 6 Siddharth Barman and Sanath Kumar Krishnamurthy. Approximation algorithms for maximin fair division. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, pages 647–664, 2017.
- 7 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 557–574, 2018.
- 8 Ben Berger, Avi Cohen, Michal Feldman, and Amos Fiat. (almost full) EFX exists for four agents (and beyond). *arXiv preprint arXiv:2102.10654*, 2021.
- 9 Vittorio Bilò, Ioannis Caragiannis, Michele Flammini, Ayumi Igarashi, Gianpiero Monaco, Dominik Peters, Cosimo Vinci, and William S. Zwicker. Almost envy-free allocations with connected bundles. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 305–322, 2018.
- 10 Sylvain Bouveret and Michel Lemaître. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems*, 30(2):259–290, 2016.
- 11 Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.
- 12 Eric Budish, Gérard P. Cachon, Judd B. Kessler, and Abraham Othman. Course match: A large-scale implementation of approximate competitive equilibrium from equal incomes for combinatorial allocation. *Operations Research*, 65(2):314–336, 2017.
- 13 Ioannis Caragiannis, Nick Gravin, and Xin Huang. Envy-freeness up to any item with high Nash welfare: The virtue of donating items. In *Proceedings of the 20th ACM Conference on Economics and Computation*, pages 527–545, 2019.
- 14 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–32, 2019.
- 15 Bhaskar Ray Chaudhury, Yun Kuen Cheung, Jugal Garg, Naveen Garg, Martin Hoefer, and Kurt Mehlhorn. On fair division for indivisible items. In *Proceedings of the 38th IARCS*

- Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–17, 2018.
- 16 Bhaskar Ray Chaudhury, Jugal Garg, and Kurt Mehlhorn. EFX exists for three agents. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC)*, pages 1–19, 2020.
 - 17 Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, Ruta Mehta, and Pranabendu Misra. Improving EFX guarantees through rainbow cycle number. *arXiv preprint arXiv:2103.01628*, 2021.
 - 18 Bhaskar Ray Chaudhury, Telikepalli Kavitha, Kurt Mehlhorn, and Alkmini Sgouritsa. A little charity guarantees almost envy-freeness. In *Proceedings of the 31st Symposium on Discrete Algorithms (SODA)*, pages 2658–2672, 2020.
 - 19 Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V Vazirani, and Sadra Yazdanbod. Convex program duality, Fisher markets, and Nash social welfare. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, pages 459–460, 2017.
 - 20 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. *SIAM Journal on Computing*, 47(3):1211–1236, 2018.
 - 21 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Approximating the Nash social welfare with budget-additive valuations. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2326–2340. SIAM, 2018.
 - 22 Jugal Garg, Pooja Kulkarni, and Rucha Kulkarni. Approximating Nash social welfare under submodular valuations through (un) matchings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, pages 2673–2687. SIAM, 2020.
 - 23 Jugal Garg and Setareh Taki. An improved approximation algorithm for maximin shares. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 379–380, 2020.
 - 24 Mohammad Ghodsi, Mohammad Taghi HajiAghayi, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods: Improvements and generalizations. In *Proceedings of the 19th ACM Conference on Economics and Computation (EC)*, pages 539–556, 2018.
 - 25 Jonathan R. Goldman and Ariel D. Procaccia. Spliddit: unleashing fair division algorithms. *ACM SIGecom Exchanges*, 13(2):41–46, 2014.
 - 26 David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *Journal of ACM (JACM)*, 65(2):1–27, 2018.
 - 27 Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible items. *Information Processing Letters*, 122:17–20, 2017.
 - 28 Wenzheng Li and Jan Vondrák. A constant-factor approximation algorithm for Nash Social Welfare with submodular valuations. *arXiv preprint arXiv:2103.10536*, 2021.
 - 29 Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 125–131, 2004.
 - 30 Ryoga Mahara. Existence of EFX for two additive valuations. *arXiv preprint arXiv:2008.08798*, 2020.
 - 31 Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. *SIAM Journal on Discrete Mathematics*, 34(2):1039–1068, 2020.
 - 32 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16(1):101–104, 1948.

FPT and FPT-Approximation Algorithms for Unsplittable Flow on Trees

Tomás Martínez-Muñoz ✉

Mathematical Engineering Department, University of Chile, Santiago, Chile

Andreas Wiese ✉

Department of Industrial Engineering, University of Chile, Santiago, Chile

Abstract

We study the unsplittable flow on trees (UFT) problem in which we are given a tree with capacities on its edges and a set of tasks, where each task is described by a path and a demand. Our goal is to select a subset of the given tasks of maximum size such that the demands of the selected tasks respect the edge capacities. The problem models throughput maximization in tree networks. The best known approximation ratio for (unweighted) UFT is $O(\log n)$. We study the problem under the angle of FPT and FPT-approximation algorithms. We prove that

- UFT is FPT if the parameters are the cardinality k of the desired solution and the number of different task demands in the input,
- UFT is FPT under $(1 + \delta)$ -resource augmentation of the edge capacities for parameters k and $1/\delta$, and
- UFT admits an FPT-5-approximation algorithm for parameter k .

One key to our results is to compute structured hitting sets of the input edges which partition the given tree into $O(k)$ clean components. This allows us to guess important properties of the optimal solution. Also, in some settings we can compute *core sets* of subsets of tasks out of which at least one task i is contained in the optimal solution. These sets have bounded size, and hence we can guess this task i easily.

A consequence of our results is that the integral multicommodity flow problem on trees is FPT if the parameter is the desired amount of sent flow. Also, even under $(1 + \delta)$ -resource augmentation UFT is APX-hard, and hence our FPT-approximation algorithm for this setting breaks this boundary.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases FPT algorithms, FPT-approximation algorithms, packing problems, unsplittable flow, trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.67

Funding *Tomás Martínez-Muñoz*: Partially supported by CMM ANID PIA AFB170001.

Andreas Wiese: Partially supported by FONDECYT Regular grant 1200173.

1 Introduction

The unsplittable flow on trees (UFT) problem is a natural problem which models throughput maximization in tree networks. We are given a tree $G = (V, E)$ where each edge e has a capacity $u(e) \in \mathbb{N}$. Also, we are given a set of tasks \mathcal{T} where each task $i \in \mathcal{T}$ is described by a path $P(i) \subseteq E$ and a demand $d(i) \in \mathbb{N}$. Our goal is to select a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ of maximum cardinality whose combined demands respect the edge capacities, i.e., $\sum_{i \in \mathcal{T}': e \in P(i)} d(i) \leq u(e)$ for each edge e . Hence, one application is that each task models a possible transmission between two nodes in a (tree) network in which the edges have bounded capacities. Note that UFT generalizes the integral multi-commodity flow on trees problem¹

¹ In the integral multi-commodity flow on trees problem we are given a tree G with edge capacities u , and additionally pairs $s_i, t_i \in V$ of source and sink vertices. The goal is to select an integral amount of



which is known to be APX-hard [19]. Also, it generalizes the well-studied unsplittable flow on path (UFP) problem which has several applications, e.g., in caching [14], scheduling [4], and bandwidth allocation [12]. For UFT there is a $O(\log n)$ -approximation algorithm [10] (and $O(\log^2 n)$ -approximation algorithms for the weighted case [10, 18] and for submodular objective functions [1]). In particular, it is open to construct a $O(1)$ -approximation algorithm for UFT.

In order to obtain better approximation ratios for combinatorial optimization problems, one can study them under the angle of FPT-approximation algorithms. In such algorithms, one defines some quantity to be a fixed parameter k which we define to be the size of the desired solution for our case of UFT, and then ensures that the running time is of the form $f(k)n^{O(1)}$ for some computable function f . Furthermore, for UFT we define that an FPT- α -approximation algorithm is an algorithm that either computes a solution of size at least k/α or asserts that there is no solution of size k . In the last years there several such results have been found for different problems. For example, for k -MEDIAN and k -MEANS there are tight FPT-approximation algorithms known with approximation ratios of $1 + 2/e + \epsilon$ and $1 + 8/e + \epsilon$, respectively [15], while the best known polynomial time algorithms for the problems have ratios of 2.611 [8] and 6.357 [3], respectively. For FACILITY LOCATION there is an FPT-approximation algorithm with a ratio of essentially $1.463 + \epsilon$ [15] which matches a known lower bound for (pure) approximation algorithms [21], while the best known upper bound is 1.488 [23]. For the capacitated versions of k -MEDIAN and k -MEANS there are FPT- $(3 + \epsilon)$ - and FPT- $(9 + \epsilon)$ -approximation algorithms known, respectively [16] while the best known polynomial time approximation ratio is only $O(\log k)$ based on an algorithm in [9] (see also [2]). For UFP there is an FPT- $(1 + \epsilon)$ -approximation algorithm [25] while the best known polynomial time approximation ratio is only $5/3 + \epsilon$ [20]. For some W[1]-hard problems, there are even FPT-approximation algorithms known whose approximation ratios beat the best possible ratios of pure approximation algorithms. For example, for the STRONGLY CONNECTED STEINER SUBGRAPH problem, there is an FPT-2-approximation algorithm known when parametrized by the number of terminals [13], but there is a lower bound of $O(\log^{2-\epsilon} n)$ for (pure) approximation algorithms [22]. We refer to the surveys by Marx [24] and Feldmann et al. [17] for more results.

Given that for the mentioned problems FPT-approximation algorithms were found with better ratios than the best known (pure) approximation algorithms, this raises the question whether this is also possible for UFT. In this paper we answer this question in the affirmative and also show that certain special cases of UFT are even FPT.

1.1 Our contribution

Our first result is that UFT admits an *exact* FPT-algorithm if the parameters are k and the number \bar{d} of different task demands in the input. Our first step is to compute a hitting set, i.e., a subset of the edges such that each input task uses at least one of them. Via a routine in [19] we construct a *structured* hitting set of size $O(k)$ or directly a solution of size k (in which case we are done). In particular, our hitting set partitions the tree into $O(k)$ clean components. Then we consider an edge e from the hitting set such that there is no edge from the hitting set underneath e . We consider all input tasks that use e but no other edge from the hitting set, let us denote them by \mathcal{T}' . Using some properties of our hitting set, we show that that in FPT time we can compute a *core set* of size $f(k, \bar{d})$ for \mathcal{T}' (for some

flow to send between each pair (s_i, t_i) , in order to maximize the total amount of sent flow.

function f) which is a set $\mathcal{T}'' \subseteq \mathcal{T}'$ such that we can assume w.l.o.g. that \mathcal{T}'' contains *all* tasks from $\mathcal{T}' \cap \text{OPT}$. Therefore, in some sense, \mathcal{T}'' forms a kernel for \mathcal{T}' (in the sense of being a smaller instance that we reduce our given instance to). This allows us to guess a task in $\mathcal{T}' \cap \text{OPT}$ in time $f(k, \bar{d})$ or we guess that $\mathcal{T}' \cap \text{OPT} = \emptyset$. In either case we make progress: either we find a task in OPT or we can delete one of the $O(k)$ edges of the hitting set. Therefore, our algorithm has only $O(k)$ iterations overall and in each of them there are only $f(k, \bar{d}) + 1$ options for our guesses.

► **Theorem 1.** *There is an FPT-algorithm for UFT for parameters k and \bar{d} with a running time of $k^{O(k \cdot \bar{d}^k)} \cdot O(n^2)$.*

Note that the above mentioned integral multi-commodity flow problem (on trees) can be modeled by UFT instances in which $d(i) = 1$ for each task i and hence $\bar{d} = 1$. Therefore, we obtain an FPT-algorithm for this problem as a by-product. In particular, note that this problem is APX-hard [19] while our algorithm computes an *optimal* solution of size k .

► **Corollary 2.** *There is an FPT-algorithm for integral multi-commodity flow on trees where the parameter k denotes the amount of sent flow.*

Using the algorithm for UFT above we construct an FPT-algorithm for the general case of UFT under $(1 + \delta)$ -resource augmentation: our algorithm either computes a solution of size k that is feasible if we increase the capacity of each edge by a factor $1 + \delta$, or we assert that there is no solution of size k for the original edge capacities. Key for this is to use the available resource augmentation to reduce the given instance to a set of smaller instances in which the input tasks have only $O(\log((k/\delta)^k))$ different demands. On each of these instances we invoke the algorithm from above and combine the obtained solutions. Due to the resource augmentation, we can ensure that this union forms a feasible solution.

► **Theorem 3.** *There is an FPT-algorithm for UFT under $(1 + \delta)$ -resource augmentation with a running time of $k^{(k/\delta)^{O(k)}} n^{O(1)}$.*

Note that our results implies an (exact) FPT-algorithm for UFP (i.e., on paths, rather than trees) under $(1 + \delta)$ -resource augmentation, which was not known before. Also, already UFP is W[1]-hard [25] for parameter k (and hence also UFT), which justifies that we use resource augmentation or the additional parameter \bar{d} . Furthermore, it establishes a distinction in comparison to (pure) approximation algorithms for UFT, since UFT is still APX-hard under resource augmentation, assuming that δ is sufficiently small.

► **Theorem 4** (implied by [19, Section 4]). *UFT is APX-hard under $(1 + \delta)$ -resource augmentation for any $\delta < 1/2$.*

Then we present an FPT-5-approximation algorithm for UFT (i.e., without resource augmentation) where the fixed-parameter is only k (and not also \bar{d}). Recall that the best known polynomial time approximation algorithm for (unweighted) UFT has an approximation ratio of only $O(\log n)$. Intuitively, let $i^* \in \text{OPT}$ be the task of smallest demand in OPT . We guess which edges of our hitting set are used by i^* . Then we select the input task i of smallest demand that uses exactly these edges of the hitting set. By an exchange argument, we show that there are seven task in OPT such that we can replace these seven (unknown) tasks in OPT by our (known) task i and still obtain a feasible solution. For proving this, we again exploit some structural properties of our hitting set. Intuitively, our task i pays for seven tasks in OPT . This yields a 7-approximation algorithm when we iterate this routine (and this argument). In order to improve the approximation ratio to 5, we guess additional properties of i^* such that we can compute a task i' which we can replace by only five tasks from OPT . This yields an FPT-5-approximation algorithm.

► **Theorem 5.** *There is an FPT-5-approximation algorithm for UFT with a running time of $k^{O(k)}n^{O(1)}$.*

1.2 Other related work

For the mentioned integral multi-commodity flow problem on trees there is a 2-approximation algorithm in the unweighted case [19] and a 4-approximation algorithm in the weighted case [11]. Under the no-bottleneck-assumption (NBA), i.e., assuming that $\max_{i \in \mathcal{T}} d(i) \leq \min_{e \in E} u(e)$ there is a 48-approximation algorithm for UFT [11]. For uniform edge capacities and bounded node degrees, there is a 3.542-approximation algorithm for UFT [6]. For UFP there is no better approximation algorithm known under uniform edge capacities (and hence neither for the NBA) than the mentioned $(5/3 + \epsilon)$ -approximation for the general case [20]. However, UFP admits a QPTAS [5, 7], i.e., a $(1 + \epsilon)$ -approximation in time $n^{(\log n)^{O(1)}}$ for any constant $\epsilon > 0$, while for UFT a QPTAS is unlikely to exist since the problem is APX-hard [19].

2 Hitting sets and structuring the tree

In all our algorithms we will use hitting sets as the backbone of our computations. We define that a *hitting set* is a set of edges $E' \subseteq E$ such that each input task uses at least one edge in E' , i.e., $P(i) \cap E' \neq \emptyset$ for each $i \in \mathcal{T}$. In this section, first we show that a result in [19] implies that in time $n^{O(1)}$ we can compute a hitting set of size at most $2k$ or directly find a solution of size k . Then, we use this as a base to compute a more structured hitting set E_{hs} of size $O(k)$. Afterwards, we use E_{hs} to partition the tree G and establish some structural properties that we can assume without loss of generality. Throughout the paper, we denote by n the number of bits in the input.

First, we invoke the mentioned result to compute an initial hitting set E' .

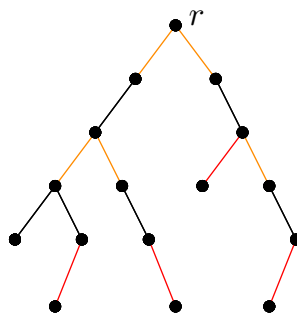
► **Lemma 6** (implicit in [19, Section 5]). *There is an algorithm with a running time of $n^{O(1)}$ that either outputs a set of tasks \mathcal{T}' with $|\mathcal{T}'| = k$ such that $P(i) \cap P(i') = \emptyset$ for all $i, i' \in \mathcal{T}'$ with $i \neq i'$, or it outputs a hitting set E' with $|E'| \leq 2k$.*

If the algorithm returns a set \mathcal{T}' of k tasks with the mentioned properties then we can simply output \mathcal{T}' since it forms a feasible solution. Assume now that the algorithm returns a hitting set E' . We want to add more edges to E' in order to obtain a more structured hitting set. For any two edges e, e' we define $\text{lca}(e, e')$ to be the vertex that is the least common ancestor of the (up to four) vertices incident to e and e' ; also, we denote by $P_{e, e'} \subseteq E$ the (unique) path in G that contains e and e' . Intuitively, a hitting set is *good* if for any two edges $e, e' \in E_{\text{hs}}$ the edges of $P_{e, e'}$ incident to $\text{lca}(e, e')$ are also in the hitting set.

► **Definition 7.** *A hitting set E_{hs} is good if for any two edges $e, e' \in E_{\text{hs}}$ we have that $\delta(\text{lca}(e, e')) \cap P_{e, e'} \subseteq E_{\text{hs}}$.*

We construct a good hitting set E_{hs} based on E' as follows. We take all edges in E' , and for any two edges $e, e' \in E'$ we add the edges in $\delta(\text{lca}(e, e')) \cap P_{e, e'}$, see Figure 1. By some standard tree arguments one can show that then $|E_{\text{hs}}| \leq 6k$. Also, E_{hs} is a good hitting set.

► **Lemma 8.** *Given a hitting set \bar{E}' , in time $n^{O(1)}$ we can construct a good hitting set \bar{E}_{hs} with $|\bar{E}_{\text{hs}}| \leq 3|\bar{E}'|$.*



■ **Figure 1** Assume that the red edges form a hitting set \bar{E}' . Then the union of the red and orange edges forms a good hitting set \bar{E}_{hs} . Given the red edges, the orange edges are selected according to the proof of Lemma 8.

2.1 Backbone and hanging trees

Let E_{hs} denote the good hitting set obtained by applying Lemma 8 to E' . We use E_{hs} in order to partition the edges of G into a backbone of *highway edges* (that will contain E_{hs}) and some subtrees which we will call *hanging trees*. First, we are interested in the edges that lie on some path that connects two edges in E_{hs} (see Figure 2). We say that an edge $e \in E$ is a *highway edge* if there are two edges $e', e'' \in E_{\text{hs}}$ such that $e \in P_{e', e''}$. Let E_{hs}^* denote the set of all highway edges and note that $E_{\text{hs}} \subseteq E_{\text{hs}}^*$. Intuitively, E_{hs}^* forms a backbone of G . Note we cannot bound $|E_{\text{hs}}^*|$ by a function of k only. On the other hand, we observe that $G[E_{\text{hs}}^*]$ is a tree. If an edge $e \in E_{\text{hs}}^*$ is incident to a leaf in $G[E_{\text{hs}}^*]$ then we say that e is a *final edge*. Observe that then $e \in E_{\text{hs}}$ and thus there are at most $6k$ final edges. The final edges will play a special role later.

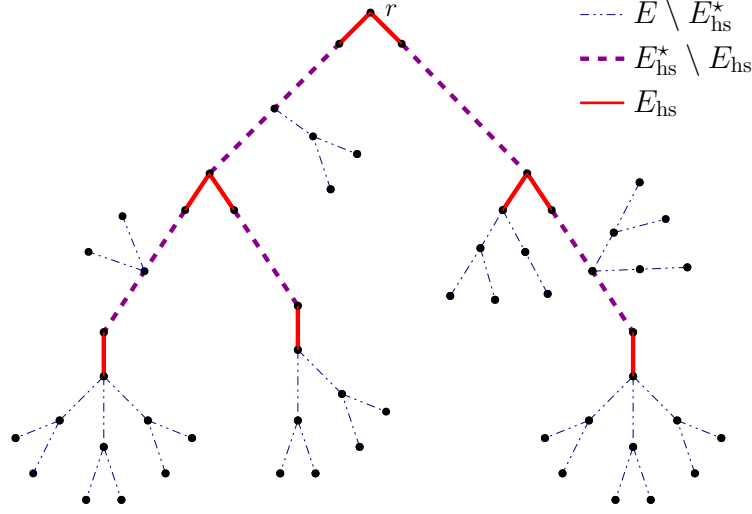
We say that the edges in $E \setminus E_{\text{hs}}^*$ are *black edges*. Note that they form trees. For each connected component H of $G[E \setminus E_{\text{hs}}^*]$ (i.e., the subgraph of G induced by $E \setminus E_{\text{hs}}^*$), we say that H forms a *hanging tree* and we define its root s to be the vertex of H that is closest to the root of G . In the sequel, we will write H_s for a hanging tree with root s . For a hanging tree H_s we say that its *depth* is the maximum distance of a vertex in H_s to s .

By some easy transformations, we can ensure that the depth of each hanging tree H_s is bounded by a function in k and $\bar{d} := |\{d_i | i \in \mathcal{T}\}|$, i.e., \bar{d} is the number of different task demands in the input. To this end, we note that if there are two edges $e, e' \in H_s$ appearing in this order on the path from s to a leaf of H_s and $u(e) \leq u(e')$, then we can contract e' since any input task T_i using e' also uses e (since $P(i)$ uses at least one edge of E_{hs} and hence $P(i)$ must pass through s). Also, we can assume w.l.o.g. that for each edge e its capacity $u(e)$ is the sum of the demands of at most k input tasks. With these ideas, we can prove the following lemma for which the intuition is that D is bounded by some parameter.

► **Lemma 9.** *In time $O(n^2)$ we can construct an equivalent instance (G', \mathcal{T}') in which the depth of every hanging tree is at most $(\bar{d} + 1)^k + 1$.*

3 Parameterized algorithm for UFT

In this section we present an FPT-algorithm for UFT, assuming that our fixed parameters are k and \bar{d} (recall that \bar{d} is the number of different task demands in the input). In particular, this shows that UFT is FPT if all tasks have unit demands (which is APX-hard).



■ **Figure 2** Partition of E , where E_{hs} are the edges in the good hitting set, E_{hs}^* are the edges in highway paths, and $E \setminus E_{hs}^*$ are the edges in hanging trees.

Our strategy is to identify a subset $\mathcal{T}' \subseteq \mathcal{T}$ of tasks for which we can compute a core set which is intuitively a subset $K \subseteq \mathcal{T}'$ for which we can assume that it contains all tasks in $\mathcal{T}' \cap \text{OPT}$ for some optimal solution OPT . We will ensure that the size of our core set is bounded by a function in k and \bar{d} . Hence, in some sense it is a kernel for the set \mathcal{T}' .

► **Definition 10.** Given a set $\mathcal{T}' \subseteq \mathcal{T}$, we say that a set $K \subseteq \mathcal{T}'$ is a core set for \mathcal{T}' if for every feasible set of tasks $\mathcal{S} \subseteq \mathcal{T}$ there exists a feasible set of tasks \mathcal{S}' such that $|\mathcal{S}'| = |\mathcal{S}|$, $\mathcal{S}' \setminus \mathcal{T}' = \mathcal{S} \setminus \mathcal{T}'$, and $\mathcal{S}' \cap \mathcal{T}' \subseteq K$.

We will ensure that $|K| \leq f(k, \bar{d})$ for our computed core sets K , for some function f . Hence, having computed K , in time $O(f(k, \bar{d}))$ we can guess a task from $\mathcal{T}' \cap \text{OPT}$ or guess that $\mathcal{T}' \cap \text{OPT} = \emptyset$. In our main algorithm, we will compute $O(k)$ core sets for different sets \mathcal{T}' , guess which of their tasks are in OPT , and in this way eventually find a solution of size k in time $(f(k, \bar{d}))^{O(k)} n^{O(1)}$.

First, we prove some basic properties of core sets.

► **Lemma 11.** Let $\mathcal{T}_1, \dots, \mathcal{T}_\ell$ be subsets of \mathcal{T} . If K_1, \dots, K_ℓ are core sets for $\mathcal{T}_1, \dots, \mathcal{T}_\ell$, respectively, then $\bigcup_{i=1}^\ell K_i$ is a core set for $\bigcup_{i=1}^\ell \mathcal{T}_i$.

► **Lemma 12.** Let \mathcal{T}' and \mathcal{T}'' with $\mathcal{T}'' \subseteq \mathcal{T}' \subseteq \mathcal{T}$. If K is a core set for \mathcal{T}'' and \mathcal{T}'' is a core set for \mathcal{T}' then K is also a core set for \mathcal{T}' .

3.1 Constructing core sets

We first present an algorithm \mathcal{A}^{kr} that computes a core set for any set of tasks \mathcal{T}' for which intuitively there are two hanging trees H, H' such that all tasks in \mathcal{T}' start in H and end in H' . Formally, the input of \mathcal{A}^{kr} consists of a path $P = P_{s,t}$ for two nodes s and t that lie in two different hanging trees H, H' , and of a value $d \in \mathbb{N}$ which is the demand of some input task. Let \mathcal{T}' denote the set of all tasks $i \in \mathcal{T}$ such that $P(i)$ contains the path P , $d(i) = d$, and each edge of $P(i) \setminus E_{hs}^*$ lies in H or H' . The algorithm \mathcal{A}^{kr} computes a core set K for \mathcal{T}' . For each node v in some hanging tree H denote by p_v the maximum distance of v to a leaf of H . Recall that by Lemma 9 we can assume that for every $v \in V$ follows that $p_v \leq (\bar{d} + 1)^k + 1$. For any two vertices v, v' denote by $P_{v,v'}$ the (unique) path from v to v' , and for each vertex v denote by $\delta(v)$ the set of edges incident to v .

The algorithm $\mathcal{A}^{\text{kr}}(P, d)$ works as follows:

1. If $p_s = p_t = 0$ then return an arbitrary subset of \mathcal{T}' of size $\min\{k, |\mathcal{T}'|\}$.
2. Otherwise, construct greedily a subset $A \subseteq \mathcal{T}'$ of tasks such that for every pair $i \neq j$ it holds that $P(i) \cap P(j) = P$: keep adding tasks to A until $|A| = 2k$ or if the property would not be fulfilled if we added an additional task $i \in \mathcal{T}' \setminus A$ to A .
3. If $|A| = 2k$ then return A .
4. If $|A| < 2k$ then let \bar{E} be the set of edges in $(\delta(s) \cup \delta(t)) \setminus P$ that are used by tasks in A .
5. For every edge $e_j \in \bar{E}$ compute $K_j = \mathcal{A}^{\text{kr}}(P \cup \{e_j\}, d)$
6. Return $K = \bigcup_{e_j \in \bar{E}} K_j$.

We want to show that $\mathcal{A}^{\text{kr}}(P, d)$ returns a core set for \mathcal{T}' . Observe that $|\bar{E}| \leq O(k)$, the recursion depth is bounded by $O((\bar{d} + 1)^k)$, and hence we output at most $k^{O((\bar{d} + 1)^k)}$ tasks in total. Intuitively, if in a recursive call it holds that $|A| < 2k$ then \bar{E} contains all edges in $(\delta(s) \cup \delta(t)) \setminus P$ that are used by tasks in \mathcal{T}' . Thus, for each task $i \in \mathcal{T}'$ there will be a recursive call $\mathcal{A}^{\text{kr}}(P \cup \{e_j\}, d)$ such that $P(i) \subseteq P \cup \{e_j\}$. On the other hand, if $|A| = 2k$ in some recursive call then A itself is a coresets for this call: if OPT contains a task $i \in \mathcal{T}' \setminus A$ then by the pigeon hole principle there must be another task $i' \in A \setminus \text{OPT}$ such that no task in OPT uses an edge in $P(i') \setminus P$ and hence we can swap i and i' in OPT, using that $d(i) = d(i')$. We formalize this in the following lemma.

► **Lemma 13.** *Let s, t be two nodes in two different hanging trees, let $d \in \mathbb{N}$, and let $\mathcal{T}' = \{i \in \mathcal{T} \mid P_{s,t} \subseteq P(i) \wedge d(i) = d\}$. Then $\mathcal{A}^{\text{kr}}(P_{s,t}, d)$ returns a core set for \mathcal{T}' of size at most $(4k)^{p_s+p_t} \cdot k$ in time $(4k)^{p_s+p_t} \cdot k \cdot O(n)$.*

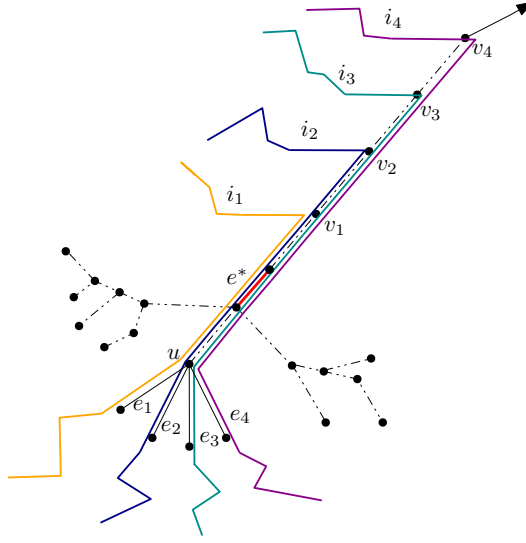
Proof. Let $\mathcal{T}_{s,t}$ be the set of tasks $i \in \mathcal{T}$ such that $P_{s,t} \subseteq P(i)$. We define the *instance depth* to be $p_s + p_t$, and we give a proof by induction on the instance depth. In the base case $p_s = p_t = 0$. Then in the first step of the algorithm two options can arise:

- $K = \mathcal{T}'$. In this case it follows directly that K is a core set for \mathcal{T}' .
- $K \neq \mathcal{T}'$. Let OPT be a feasible solution of size k . In this case it follows that K has size $2k$, and then we can replace all the tasks in $\text{OPT} \cap \mathcal{T}'$ by tasks of K .

Now consider the case that $p_s + p_t > 0$. Given an edge $e_j \in \bar{E}$, let s_j and t_j be the end nodes of the path $P \cup \{e_j\}$. It follows that $\mathcal{T}' \cap \mathcal{T}_{s_j, t_j}$ is the set of tasks in \mathcal{T}' that uses e_j . Since $|P_{s_j, t_j}| = |P_{s,t}| + 1$, then $p_{s_j} + p_{t_j} = p_s + p_t - 1$, implying by the inductive hypothesis that K_j is a core set of $\mathcal{T}' \cap \mathcal{T}_{s_j, t_j}$. We define $\mathcal{T}'' = \bigcup_{j \in \{1, \dots, |\bar{E}|\}} \mathcal{T}' \cap \mathcal{T}_{s_j, t_j}$. Lemma 11 implies that K is a core set for \mathcal{T}'' . It remains to show that \mathcal{T}'' is a core set for \mathcal{T}' . If $|A| < 2k$ then each task in \mathcal{T}' uses an edge from \bar{E} , which implies that $\mathcal{T}' = \mathcal{T}''$, so we conclude that \mathcal{T}'' is a core set for \mathcal{T}' . Since K is a core set for \mathcal{T}'' and \mathcal{T}'' is a core set for \mathcal{T}' , the Lemma 12 implies that K is a core set of \mathcal{T}' .

Assume now that $|A| = 2k$. Let OPT be a feasible solution of size k , such that among all solutions OPT' of size k with $\text{OPT} \setminus \mathcal{T}' \subseteq \text{OPT}'$, the solution OPT is the solution that maximizes $|\text{OPT}' \cap A|$. Let us assume that there is a task $i \in (\text{OPT} \cap \mathcal{T}') \setminus A$. Each task in OPT shares an edge in $E \setminus P$ with at most two tasks from A . Therefore, there are at most $2(k - 1)$ tasks $i' \in A$ such that $P(i') \setminus P$ contains an edge that is used by some task in OPT. Therefore, there exists a task $i'' \in A$ such that no edge in $P(i'') \setminus P$ is used by any tasks in OPT. Therefore, $\text{OPT} \cup \{i''\} \setminus \{i\}$ is a feasible solution and $|(\text{OPT} \cup \{i''\} \setminus \{i\}) \cap A| > |\text{OPT} \cap A|$ which is a contradiction. Therefore, A is a core set for \mathcal{T}' .

Now we analyse the running time of \mathcal{A}^{kr} . Each of the recursive calls generated by \mathcal{A}^{kr} is associated to a path P_{s_i, t_i} such that $|P_{s_i, t_i}| = |P_{s,t}| + 1$, and therefore $p_{s_i} + p_{t_i} \leq p_s + p_t - 1$. Therefore, the recursion depth is at most $p_s + p_t$. In the recursion tree each node has at most



■ **Figure 3** The sets $A = \{i_1, i_2, i_3, i_4\}$, $\bar{V} = \{v_1, v_2, v_3, v_4\}$, and $\bar{E} = \{e_1, e_2, e_3, e_4\}$ as they are defined in algorithm $\mathcal{A}^{\text{fin}}(P, d)$.

$4k$ childrens. If in a leaf of the recursion tree a core set is returned, this core set contains at most k tasks. We conclude that K is a core set of size at most $(4k)^{p_s+p_t} \cdot k$ which we compute in time $(4k)^{p_s+p_t} \cdot k \cdot O(n)$. ◀

We will use \mathcal{A}^{kr} as a subroutine in a different algorithm \mathcal{A}^{fin} for computing core sets which we describe now. Let e^* be a final edge. Let $H(e^*)$ be the hanging tree underneath e^* , i.e., that is rooted at the vertex incident to e^* that is further away from the root. The input of \mathcal{A}^{fin} consists of a path $P \subseteq H(e^*) \cup \{e^*\}$ and of a value $d \in \mathbb{N}$ which is intuitively the demand of some input task. Let \mathcal{T}' denote the set of all tasks i such that $P \subseteq P(i)$, $P(i) \cap E_{\text{hs}} = \{e^*\}$ and $d(i) = d$. The algorithm \mathcal{A}^{fin} will compute a core set for \mathcal{T}' . In order to present the algorithm, we introduce some definitions. We say that a task $i \in \mathcal{T}'$ *turns* at a node v if v is the endnode of the path $P(i) \cap E_{\text{hs}}^*$ that is closest to the root. For each task i we denote by $v(i)$ the vertex at which i turns. We define that the *level* $\ell(i)$ of a task $i \in \mathcal{T}'$ is the distance between $v(i)$ and e^* , i.e., the number of edges of the path that connects $v(i)$ with the vertex incident to e^* that is closer to the root.

Our algorithm $\mathcal{A}^{\text{fin}}(P, d)$ works as follows (see Figure 3 for an illustration):

1. Initialize $A = \emptyset$. Consider tasks in \mathcal{T}' ordered non-decreasingly by their levels, add each task $i \in \mathcal{T}'$ to A if for each previously added task $i' \in A$ it holds that $P(i) \cap P(i') \subseteq E_{\text{hs}}^* \cup P$. Stop if $|A| = 2k$ or if no more task in \mathcal{T}' can be added to A .
2. Let $\bar{V} = \{v_1, \dots, v_{|\bar{V}|}\}$ be the vertices at which the tasks in A turn, u be the endvertex of P that is in $H(e^*)$, and \bar{E} be the set of edges in $\delta(u) \setminus P$ that are used by tasks in A .
3. For every vertex $v_j \in \bar{V}$ we compute $K_j^{\text{kr}} = \mathcal{A}^{\text{kr}}(P_{v_j, u}, d)$.
4. For every edge $e_j \in \bar{E}$ we define $P_j = P \cup \{e_j\}$ and compute $K_j^{\text{fin}} = \mathcal{A}^{\text{fin}}(P_j, d)$.
5. Return

$$K = \left(\bigcup_{j \in \{1, \dots, |\bar{V}|\}} K_j^{\text{kr}} \right) \cup \left(\bigcup_{j \in \{1, \dots, |\bar{E}|\}} K_j^{\text{fin}} \right).$$

We want to show that $\mathcal{A}^{\text{fin}}(P, d)$ computes a core set with bounded size for \mathcal{T}' . If $|A| = 2k$ we can show by an exchange argument that there is an optimal solution OPT such that each task in $\text{OPT} \cap \mathcal{T}'$ turns at some vertex in V' or use some edge in E' . If $|A| < 2k$ one can show that this is automatically satisfied. In the calls to \mathcal{A}^{kr} we obtain core sets that together contain all tasks in $\text{OPT} \cap \mathcal{T}'$ that turn at some vertex in V' . The recursive calls to \mathcal{A}^{fin} compute core sets that together contain all tasks in $\text{OPT} \cap \mathcal{T}'$ that use some edge in E' . Observe that for \mathcal{A}^{fin} the tasks are ordered, favoring tasks of lower level, since those intuitively use less edges on E_{hs}^* and thus intersect with fewer other tasks on these edges. We define p_{\max} to be the maximum value p_u over all nodes $v \in V$. Since due to Lemma 9 we can assume that $p_v \leq (\bar{d} + 1)^k + 1$ for every $v \in V$, we obtain that $p_{\max} \leq (\bar{d} + 1)^k + 1$.

► **Lemma 14.** *Let e^* be a final edge, let $P \subseteq H(e^*) \cup \{e^*\}$ be a path, and let \mathcal{T}' be the set of all tasks i such that $P \subseteq P(i)$, $P(i) \cap E_{\text{hs}} = \{e^*\}$, and $d(i) = d$. The algorithm $\mathcal{A}^{\text{fin}}(P, d)$ computes a core set for \mathcal{T}' of size at most $(4k)^{2p_{\max}} \cdot k$ in time $(4k)^{2p_{\max}} \cdot k^2 \cdot O(n)$.*

3.2 UFT Algorithm

Now we describe our main FPT-algorithm for UFT which will use $\mathcal{A}^{\text{fin}}(e, d)$ as a subroutine. Recall that there are at most $6k$ edges in E_{hs} . We take an arbitrary final edge $e \in E_{\text{hs}}$. We guess whether there is a task $i \in \text{OPT}$ that uses e but no other edge in E_{hs} , i.e., such that $P(i) \cap E_{\text{hs}} = \{e\}$. If yes, we guess $d(i)$ for which there are only \bar{d} options. Then we call $\mathcal{A}^{\text{fin}}(e, d(i))$ and obtain a core set K of size at most $(4k)^{2p_{\max}} k^2$. Assume w.l.o.g. that OPT is an optimal solution with the property that K contains all tasks in OPT that use e but no other edge in E_{hs} (using that K is a core set). In time $(4k)^{2p_{\max}} k^2$ we guess a task $i \in K \cap \text{OPT}$. We add i to our computed solution and remove i from the set of input tasks \mathcal{T} . We update the edges capacities, taking into account that we selected i , i.e., we update $u(e) := u(e) - d(i)$ for each edge $e \in P(i)$. At this point, we remove from \mathcal{T} each task $i' \in \mathcal{T}$ such that $u(e) < d(i')$ for some edge $e \in P(i)$. We keep on guessing whether there is a task $i \in \text{OPT}$ that uses e but no other edge in E_{hs} , i.e., $P(i) \cap E_{\text{hs}} = \{e\}$. If yes, we repeat the process above. Otherwise, we know that there is no more task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$. Then we remove e from E_{hs} , we remove from \mathcal{T} all input tasks i with $P(i) \cap E_{\text{hs}} = \{e\}$, and we define E_{hs}^* newly based on the changed set E_{hs} . Then we apply Lemma 9 to the resulting instance.

We repeat the above process with a final edge e' in the (changed) set E_{hs} (note that possibly e' was not a final edge in the original set E_{hs}). We continue until we selected k tasks in total. If all our guesses were correct, then \mathcal{S} forms a feasible solution. Moreover, there are at most $O(k)$ guesses in total with at most $\bar{d} \cdot (4k)^{2p_{\max}} k^2$ possibilities for each guess: if we guess that there is a task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$ then subsequently we select such a task and this happens at most k times. On the other hand, if we guess that there is no task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$ then subsequently we remove e from E_{hs} and the initial set E_{hs} has only $O(k)$ edges. For each guess there are at most $\bar{d} \cdot (4k)^{2p_{\max}} k^2$ options. Hence, there are only $(\bar{d} \cdot (4k)^{2p_{\max}} k^2)^{O(k)}$ possibilities for all our guesses overall. Therefore, we obtain a running time of $k^{O(k \cdot \bar{d}^k)} \cdot O(n^2)$ overall, which yields the proof of Theorem 1.

4 Resource augmentation

In this section we present an FPT-algorithm for UFT under $(1 + \delta)$ -resource augmentation. Formally, we present an algorithm with a running time of the form $f(k, \delta)n^{O(1)}$ for some computable function f that outputs a set $\mathcal{S} \subseteq \mathcal{T}$ with $|\mathcal{S}| = k$ that is feasible under $(1 + \delta)$ -resource augmentation, i.e., $\sum_{i: i \in \mathcal{S} \cap \mathcal{T}_e} d(i) \leq (1 + \delta)u(e)$ for each $e \in E$, or outputs that there is no solution of size k (for the original edge capacities).

Our strategy is to split the input tasks \mathcal{T} into groups such that the demands of any two tasks in different groups differ at least by a factor k/δ . Then we can compute a solution for each group separately and their union is a (global) solution under resource augmentation, as the following lemma shows.

► **Lemma 15.** *Consider sets of tasks $\{\mathcal{T}_j\}_{j \in \mathbb{N}}$ with $\mathcal{T}_j \subseteq \mathcal{T}$ for each $j \in \mathbb{N}$ such that for each $j, j' \in \mathbb{N}$ with $j \neq j'$ and any two tasks $T_i \in \mathcal{T}_j$, $T_{i'} \in \mathcal{T}_{j'}$ it holds that $d(i) > \frac{k}{\delta} d(i')$ or $d(i') > \frac{k}{\delta} d(i)$. For each $j \in \mathbb{N}$ let $\mathcal{S}_j \subseteq \mathcal{T}_j$ be a solution with at most k tasks that is feasible under $(1 + \delta')$ -resource augmentation for some $\delta' \geq 0$, and suppose that $\delta < 1/2$. Then $\bigcup_{j \in \mathbb{N}} \mathcal{S}_j$ is feasible under $(1 + 2\delta' + 2\delta)$ -resource augmentation.*

On the other hand, if we can ensure that within each group the task demands differ by at most a factor $(k/\delta)^k$, then we can compute a solution of size k for this group that is feasible under $(1 + \delta)$ -resource augmentation as follows: we first round the task demands to powers of $1 + \delta$ and then invoke our algorithm due to Theorem 1, using that the number of different task demands is only $\log_{1+\delta}((k/\delta)^k)$.

► **Lemma 16.** *Consider a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ such that for any two tasks $i, i' \in \mathcal{T}'$ it holds that $(\frac{\delta}{k})^k d(i) \leq d(i') \leq (\frac{k}{\delta})^k d(i)$. Then in time $k^{O(k^{k+2} \cdot (\log_{1+\delta}(k/\delta))^{k+1})} \cdot O(n^2)$ we can compute for any $k' \leq k$ a solution $\mathcal{S}' \subseteq \mathcal{T}'$ of size k' that is feasible under $(1 + \delta)$ -resource augmentation, or assert that there is no solution of size k' (for the original edge capacities).*

Now with a shifting argument we can guess sets of tasks $\{\mathcal{T}_j\}_{j \in \mathbb{N}}$ that satisfy the condition of Lemma 15, such that each set \mathcal{T}_j satisfies the condition of Lemma 16, and additionally $\bigcup_{j \in \mathbb{N}} \mathcal{T}_j$ contains a solution of size k . We apply Lemma 16 to each set \mathcal{T}_j to find the largest $k' \leq k$ for which there exists a solution, denote by \mathcal{S}_j the largest solution found, and output the union $\bigcup_{j \in \mathbb{N}} \mathcal{S}_j$ which is feasible under $(1 + 4\delta)$ -resource augmentation due to Lemma 15.

► **Theorem 17.** *There is an FPT-algorithm for UFT under $(1 + \delta)$ -resource augmentation with a running time of $k^{O((k \cdot \log_{1+\delta}(k/\delta))^{k+2})} \cdot O(n^2)$.*

5 FPT-approximation algorithm

In this section we present an FPT-5-approximation algorithm for UFT, i.e., given a parameter k , in time $f(k)n^{O(1)}$ our algorithm either finds a solution of size $k/5$ or asserts that there are no solution of size k . First, we present a simpler FPT-7-approximation algorithm and then explain how to improve it to an FPT-5-approximation.

Again, we invoke Lemmas 6 and 8 to construct a good hitting set E_{hs} (or directly obtain a solution of size k). Since $|E_{\text{hs}}| = O(k)$, we observe that the graph $(V, E \setminus E_{\text{hs}})$ has $K = O(k)$ connected components; we denote them by G_1, \dots, G_K . Since E_{hs} is a hitting set, we obtain the following lemma.

► **Lemma 18.** *For each task $i \in \mathcal{T}$ the two endvertices of $P(i)$ lie in two different components in $\{G_1, \dots, G_K\}$.*

Let $i_1^* \in \text{OPT}$ denote the task with smallest demand in OPT. We guess the two components of the two endvertices of $P(i_1^*)$. We select the input task i_1 with smallest demand whose path has its endvertices in the same two components. It might be that $i_1 \neq i_1^*$; however, we will show that we can find seven tasks $i'_1, \dots, i'_7 \in \text{OPT}$ such that if we replace i'_1, \dots, i'_7 by i_1 then we still obtain a feasible solution. Intuitively, the task i_1 (which we select) pays for the tasks i'_1, \dots, i'_7 (which we lose). We continue iteratively until we picked $\lceil k/7 \rceil$ tasks.

Formally, our algorithm runs in $\lceil k/7 \rceil$ rounds where in each round we select one task i . Let $\text{OPT}_0 = \text{OPT}$. Suppose that after $k' \in \{0, \dots, \lceil k/7 \rceil - 1\}$ rounds we selected tasks $i_1, \dots, i_{k'}$ and defined a solution $\text{OPT}_{k'} \subseteq \text{OPT}$ such that $\text{OPT}_{k'} \cup \{i_1, \dots, i_{k'}\}$ is feasible (note that this is clearly true for $k' = 0$). Let $i_{k'+1}^* \in \text{OPT}_{k'}$ denote a task with smallest demand in $\text{OPT}_{k'}$, i.e., such that $d(i_{k'+1}^*) \leq d(i)$ for each $i \in \text{OPT}_{k'}$. We guess the two components from $\{G_1, \dots, G_K\}$ that contain the two endvertices of $P(i_{k'+1}^*)$. Let $i_{k'+1}$ denote the input task i of smallest demand with the property that $P(i)$ has its endvertices in the same components and $\{i, i_1, \dots, i_{k'}\}$ forms a feasible solution. We select $i_{k'+1}$ and prove in the next lemma that intuitively we can select $i_{k'+1}$ if we are willing to sacrifice at most seven tasks from $\text{OPT}_{k'}$.

► **Lemma 19.** *There exist tasks $i'_1, \dots, i'_7 \in \text{OPT}_{k'}$ such that $\text{OPT}_{k'} \setminus \{i'_1, \dots, i'_7\} \cup \{i_1, \dots, i_{k'}, i_{k'+1}\}$ is feasible.*

Proof. Let G' and G'' denote the two components from $\{G_1, \dots, G_K\}$ that contain the two endvertices of $P(i_{k'+1}^*)$. The idea behind the proof is to identify seven tasks $i'_1, \dots, i'_7 \in \text{OPT}_{k'}$ such that $d(i'_\ell) \geq d(i_{k'+1}^*) \geq d(i_{k'+1})$ for each $\ell \in \{1, \dots, 7\}$ and such that the edges of their paths contain all the edges of $P(i_{k'+1})$ that are used by tasks in $\text{OPT}_{k'}$ (i.e., edges on which selecting $i_{k'+1}$ potentially causes conflicts with other tasks in $\text{OPT}_{k'}$). We will select three tasks for the edges in $P(i_{k'+1}) \cap E(G')$, three for the edges in $P(i_{k'+1}) \cap E(G'')$, and one for the edges in $P(i_{k'+1}) \setminus (E(G') \cup E(G''))$.

Consider the edges in $P(i_{k'+1}) \cap E(G')$. We partition them into $P(i_{k'+1}) \cap E(G') \setminus E_{\text{hs}}^*$ and $P(i_{k'+1}) \cap E(G') \cap E_{\text{hs}}^*$. If an edge $e \in P(i_{k'+1}) \cap E(G') \setminus E_{\text{hs}}^*$ is used by some task $i \in \text{OPT}_{k'}$, then e is also used by the task $i' \in \text{OPT}_{k'}$ that maximizes $|P(i') \cap P(i_{k'+1}) \cap E(G') \setminus E_{\text{hs}}^*|$. Let i'_1 denote this task i' . Regarding the edges in $P(i_{k'+1}) \cap E(G') \cap E_{\text{hs}}^*$, note that they form a path, and let u and v be its end vertices. Let i'_2 denote the task $i \in \text{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(G') \cap E_{\text{hs}}^*|$. Similarly, let i'_3 denote the task $i \in \text{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(G') \cap E_{\text{hs}}^*|$. Then, if some task $i \in \text{OPT}_{k'}$ uses an edge $e \in P(i_{k'+1}) \cap E(G') \cap E_{\text{hs}}^*$ then $e \in P(i'_2)$ or $e \in P(i'_3)$ (see Figure 4). In a similar way, we identify three tasks i'_4, i'_5 and i'_6 for $P(i_{k'+1}) \cap E(G'')$.

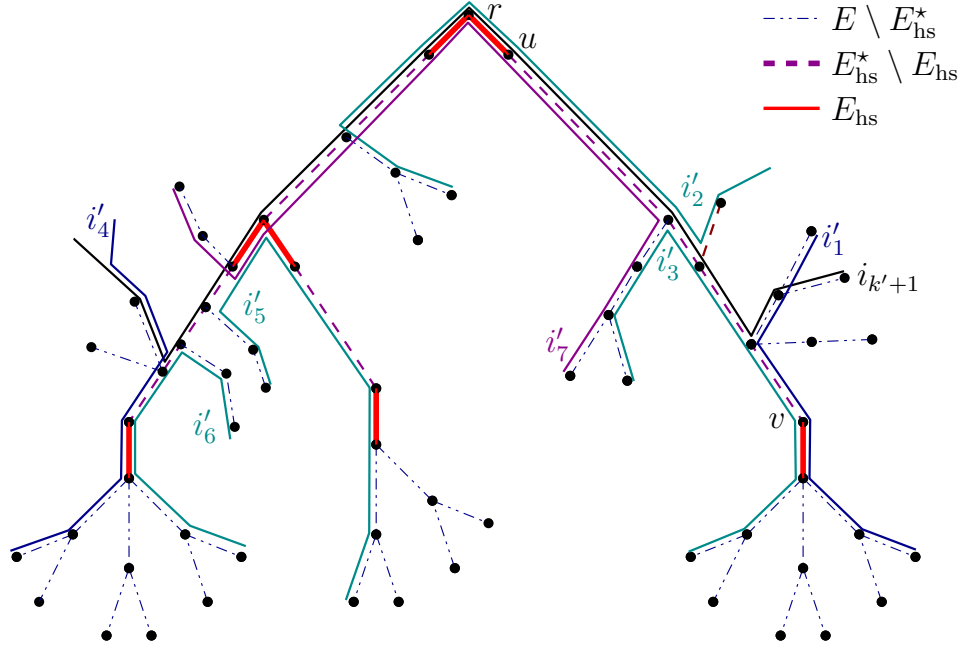
Finally, all the edges in $P(i_{k'+1}) \setminus (G' \cup G'')$ are used by $i_{k'+1}^*$, and we define $i'_7 := i_{k'+1}^*$. If an edge e is used by some task $i \in \text{OPT}_{k'}$, then $e \in P(i'_\ell)$ for some $\ell \in \{1, \dots, 7\}$. Since $d(i'_\ell) \geq d(i_{k'+1}^*) \geq d(i_{k'+1})$ for each $\ell \in \{1, \dots, 7\}$, the tasks i'_1, \dots, i'_7 satisfy the claim of the lemma. ◀

We remark that the tasks i'_1, \dots, i'_7 might not be pairwise distinct. We define $\text{OPT}_{k'+1} := \text{OPT}_{k'} \setminus \{i'_1, \dots, i'_7\}$ and iterate. Since in each iteration we picked one task and removed at most seven tasks from OPT , one can show easily that at the end we select $\lceil k/7 \rceil$ tasks, assuming that $|\text{OPT}| \geq k$. Our running time is $k^{O(k)} n^{O(1)}$ since in each of the k iterations there are $O(k^2)$ options for our guesses.

► **Theorem 20.** *There is an FPT-7-approximation algorithm for UFT with a running time of $k^{O(k)} n^{O(1)}$.*

5.1 Improvement to an FPT-5-approximation

We improve our approximation factor to 5 by doing additional guesses when we select the task $i_{k'}$ in each round k' . Assume again that at the beginning of round $k' + 1$ we have already selected tasks $i_1, \dots, i_{k'}$, and defined a set $\text{OPT}_{k'} \subseteq \text{OPT}$ such that $\text{OPT}_{k'} \cup \{i_1, \dots, i_{k'}\}$ is feasible. Like before, let $i_{k'+1}^* \in \text{OPT}_{k'}$ denote the task with smallest demand in $\text{OPT}_{k'}$ and we guess the components from G_1, \dots, G_K that contain the endvertices of $P(i_{k'+1}^*)$. Let



■ **Figure 4** The task $i_{k'+1}$ is selected, and Lemma 19 shows that if we remove the tasks i'_1, \dots, i'_7 from $\text{OPT}_{k'}$, we can add the task $i_{k'+1}$ to $\text{OPT}_{k'}$ since each edge on $P(i_{k'+1})$ is used by one of the tasks i'_1, \dots, i'_7 or by no task from $\text{OPT}_{k'}$.

G' and G'' be these components. Like before, let $i_{k'+1}$ denote the input task i of smallest demand with the property that $P(i)$ has its endvertices in G' and G'' and $\{i, i_1, \dots, i_{k'}\}$ forms a feasible solution. We do not select $i_{k'+1}$ yet; instead, we guess some properties of $P(i_{k'+1}^*)$, more precisely, of the part of $P(i_{k'+1}^*)$ within G' and G'' . Since E_{hs} is a good hitting set, it holds that G' (or G'') restricted to E_{hs}^* is a path.

► **Lemma 21.** *Let $\tilde{G} \in \{G_1, \dots, G_K\}$. Then $G[E_{\text{hs}}^* \cap E(\tilde{G})]$ is a path (possibly with zero edges).*

Let $\bar{G}' := G[E_{\text{hs}}^* \cap E(G')]$ and $\bar{G}'' := G[E_{\text{hs}}^* \cap E(G'')]$. Observe that $P(i_{k'+1}^*)$ uses some edges of \bar{G}' and also $P(i_{k'+1})$ uses some edges of \bar{G}' . Note that the respective sets of edges are contained in each other. The same holds for \bar{G}'' . It turns out that for Lemma 19 a particularly bad case is when $P(i_{k'+1})$ uses strictly more edges of both \bar{G}' and \bar{G}'' than $P(i_{k'+1}^*)$, i.e., then we need to remove seven tasks from $\text{OPT}_{k'}$, rather than fewer tasks. Therefore, we guess whether this bad case applies. If yes, instead of $i_{k'+1} =: i_{k'+1}^{(1)}$ we consider a task $i_{k'+1}^{(2)}$ which is the input task i of smallest demand with endvertices in G' and G'' , such that $\{i, i_1, \dots, i_{k'}\}$ forms a feasible solution and $P(i)$ uses strictly less edges than $P(i_{k'+1}^{(1)})$ of both \bar{G}' and \bar{G}'' . Again, we guess whether the bad case applies for $i_{k'+1}^{(2)}$. If yes, we replace $i_{k'+1}^{(2)}$ by some task $i_{k'+1}^{(3)}$ that uses even fewer edges of \bar{G}' and \bar{G}'' and so on. We repeat this for at most $2k$ iterations. Formally, if for some $\ell \in \{1, \dots, 2k\}$ we defined the task $i_{k'+1}^{(\ell)}$ then we guess whether the bad case applies for $i_{k'+1}^{(\ell)}$, i.e., whether $P(i_{k'+1}^{(\ell)})$ uses strictly more edges of both \bar{G}' and \bar{G}'' than $P(i_{k'+1}^*)$. If yes, we define the task $i_{k'+1}^{(\ell+1)}$ to be the input task i of smallest demand with endvertices in G' and G'' , such that $\{i, i_1, \dots, i_{k'}\}$ forms a feasible solution and $P(i)$ uses strictly less edges than $P(i_{k'+1}^{(\ell)})$ of both \bar{G}' and \bar{G}'' . If for some $i_{k'+1}^{(\ell)}$ with $\ell \in \{1, \dots, 2k\}$ the bad case does not apply then intuitively we can show that

for adding $i_{k'+1}^{(\ell)}$ we need to remove only five tasks from OPT, rather than seven. In this case we select $i_k := i_{k'+1}^{(\ell)}$ and define $\text{OPT}_{k'+1} := \text{OPT}_{k'} \setminus \{i'_1, \dots, i'_5\}$ for the tasks i'_1, \dots, i'_5 due to the following lemma, and continue with the next round $k' + 2$.

► **Lemma 22.** *Suppose that for some $\ell \in \{1, \dots, 2k\}$ it holds that $P(i_{k'+1}^{(\ell)})$ does not use strictly more edges of both \bar{G}' and \bar{G}'' than $P(i_{k'+1}^*)$. Then there exist tasks $i'_1, \dots, i'_5 \in \text{OPT}_{k'}$ such that $\text{OPT}_{k'} \setminus \{i'_1, \dots, i'_5\} \cup \{i_1, \dots, i_{k'}, i_{k'+1}^{(\ell)}\}$ is feasible.*

Proof. Assume w.l.o.g. that $P(i_{k'+1}^{(\ell)})$ does not use strictly more edges of \bar{G}'' than $P(i_{k'+1}^*)$. Following the idea behind Lemma 19 we identify five tasks $i'_1, \dots, i'_5 \in \text{OPT}_{k'}$ such that $d(i'_j) \geq d(i_{k'+1}^*) \geq d(i_{k'+1}^{(\ell)})$ for each $j \in \{1, \dots, 5\}$ and such that the edges of their paths (together) contain all the edges of $P(i_{k'+1}^{(\ell)})$ that are used by tasks in $\text{OPT}_{k'}$ (i.e., edges on which selecting $i_{k'+1}^{(\ell)}$ potentially causes conflicts with other tasks in $\text{OPT}_{k'}$). We will select three tasks for the edges in $P(i_{k'+1}^{(\ell)}) \cap E(G')$, one for the edges in $P(i_{k'+1}^{(\ell)}) \cap E(G'') \setminus E(\bar{G}'')$, and one for the edges in $P(i_{k'+1}^{(\ell)}) \setminus (E(G') \cup E(G''))$ and the edges in $P(i_{k'+1}^{(\ell)}) \cap E(\bar{G}'')$.

Consider the edges in $P(i_{k'+1}^{(\ell)}) \cap E(G')$. We partition them into $P(i_{k'+1}^{(\ell)}) \cap E(G') \setminus E(\bar{G}')$ and $P(i_{k'+1}^{(\ell)}) \cap E(\bar{G}')$. If an edge $e \in P(i_{k'+1}^{(\ell)}) \cap E(G') \setminus E(\bar{G}')$ is used by some task $i \in \text{OPT}_{k'}$, then e is also used by the task $i' \in \text{OPT}_{k'}$ that maximizes $|P(i') \cap P(i_{k'+1}^{(\ell)}) \cap E(G') \setminus E(\bar{G}')|$. Let i'_1 denote this task i' . Regarding the edges in $P(i_{k'+1}^{(\ell)}) \cap E(\bar{G}')$, note that they form a path, and let u and v be its end vertices. Let i'_2 denote the task $i \in \text{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}')|$. Similarly, let i'_3 denote the task $i \in \text{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}')|$. Then, if some task $i \in \text{OPT}_{k'}$ uses an edge $e \in P(i_{k'+1}^{(\ell)}) \cap E(\bar{G}')$ then $e \in P(i'_2)$ or $e \in P(i'_3)$.

Regarding the edges in $P(i_{k'+1}^{(\ell)}) \cap E(G'') \setminus E(\bar{G}'')$, if an edge $e \in P(i_{k'+1}^{(\ell)}) \cap E(G'') \setminus E(\bar{G}'')$ is used by some task $i \in \text{OPT}_{k'}$, then e is also used by the task $i' \in \text{OPT}_{k'}$ that maximizes $|P(i') \cap P(i_{k'+1}^{(\ell)}) \cap E(G'') \setminus E(\bar{G}'')|$. Let i'_4 denote this task i' . Regarding the edges in $P(i_{k'+1}^{(\ell)}) \cap E(\bar{G}'')$ and the edges in $P(i_{k'+1}^{(\ell)}) \setminus (E(G') \cup E(G''))$, they are all used by $i_{k'+1}^*$, which we now identify as i'_5 .

It follows that if an edge e is used by some task $i \in \text{OPT}_{k'}$, then $e \in P(i'_j)$ for some $j \in \{1, \dots, 5\}$. Since $d(i'_j) \geq d(i_{k'+1}^*) \geq d(i_{k'+1}^{(\ell)})$ for each $j \in \{1, \dots, 5\}$, the tasks i'_1, \dots, i'_5 satisfy the claim of the lemma. ◀

On the other hand, if the bad case applied to each $i_{k'+1}^{(\ell)}$ with $\ell \in \{1, \dots, 2k\}$ then we can show that there must be one task $i_{k'+1}^{(\ell^*)}$ with $\ell^* \in \{1, \dots, 2k\}$ such that the edges of $P(i_{k'+1}^{(\ell^*)}) \setminus E_{\text{hs}}^*$ are not used by any task in OPT. It turns out that this is again a good case. Intuitively, then we do not need to remove any task from OPT in order to make space for $i_{k'+1}^{(\ell^*)}$ on the edges in $P(i_{k'+1}^{(\ell^*)}) \setminus E_{\text{hs}}^*$ and, therefore, it turns out that we need to remove only five tasks from OPT. In this case we guess ℓ^* , select $i_k := i_{k'+1}^{(\ell^*)}$ and define $\text{OPT}_{k'+1} := \text{OPT}_{k'} \setminus \{i'_1, \dots, i'_5\}$ for the tasks i'_1, \dots, i'_5 due to the following lemma, and continue with the next round $k' + 2$.

► **Lemma 23.** *Suppose that for each $\ell \in \{1, \dots, 2k\}$ it holds that $P(i_{k'+1}^{(\ell)})$ uses strictly more edges of \bar{G}' and \bar{G}'' than $P(i_{k'+1}^*)$. Then there is an $\ell^* \in \{1, \dots, 2k\}$ for which there exist tasks $i'_1, \dots, i'_5 \in \text{OPT}_{k'}$ such that $\text{OPT}_{k'} \setminus \{i'_1, \dots, i'_5\} \cup \{i_1, \dots, i_{k'}, i_{k'+1}^{(\ell^*)}\}$ is feasible.*

Proof. Since for each $\ell \in \{1, \dots, 2k\}$ it holds that $P(i_{k'+1}^{(\ell)})$ uses strictly more edges of \bar{G}' and \bar{G}'' than $P(i_{k'+1}^*)$, then there is an $\ell^* \in \{1, \dots, 2k\}$ such that $|P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}') \setminus E_{\text{hs}}^*| = |P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}'') \setminus E_{\text{hs}}^*| = 0$. Following the idea behind Lemma 19 we identify five tasks

$i'_1, \dots, i'_5 \in \text{OPT}_{k'}$ such that $d(i'_{\ell^*}) \geq d(i'_{k'+1}) \geq d(i_{k'+1})$ for each $j \in \{1, \dots, 5\}$ and such that the edges of their paths contain all the edges of $P(i_{k'+1})$ that are used by tasks in $\text{OPT}_{k'}$ (i.e., edges on which selecting $i_{k'+1}$ potentially causes conflicts with other tasks in $\text{OPT}_{k'}$). Now will select two tasks for the edges in $P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}')$, two for the ones in that in $P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}'')$ and one for the edges in $P(i_{k'+1}^{(\ell^*)}) \setminus (E(\bar{G}') \cup E(\bar{G}''))$.

Consider the edges in $P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}'')$. We partition them into $P(i_{k'+1}^{(\ell^*)}) \cap E(G'') \setminus E(\bar{G}'')$ and $P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}'')$. There is no edge $P(i_{k'+1}^{(\ell^*)}) \cap E(G'') \setminus E(\bar{G}'')$ used by some task $i \in \text{OPT}_{k'}$, so we don't need to define a task to cover those edges. Regarding the edges in $P(i_{k'+1}^{(\ell^*)}) \cap E(\bar{G}'')$, we note that they form a path, and let u and v be their end vertices. Let i'_1 denote the task $i \in \text{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}') \cap E_{\text{hs}}^*|$. Similarly, let i'_2 denote the task $i \in \text{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}'') \cap E_{\text{hs}}^*|$. Then, if some task $i \in \text{OPT}_{k'}$ uses an edge $e \in P(i_{k'+1}) \cap E(\bar{G}'')$ then $e \in P(i'_1)$ or $e \in P(i'_2)$. In a similar way, we identify two tasks i'_3 and i'_4 for $P(i_{k'+1}^{(\ell^*)}) \cap E(G'')$.

Finally, all the edges in $P(i_{k'+1}^{(\ell^*)}) \setminus (G' \cup G'')$ are used by $i_{k'+1}^*$, and we define $i'_5 := i_{k'+1}^*$. It follows that if an edge e is used by some task $i \in \text{OPT}_{k'}$, then $e \in P(i'_j)$ for some $j \in \{1, \dots, 5\}$. Since $d(i'_j) \geq d(i_{k'+1}^*) \geq d(i_{k'+1})$ for each $j \in \{1, \dots, 5\}$, the tasks i'_1, \dots, i'_5 satisfy the claim of the lemma. \blacktriangleleft

Since in each iteration we picked one task and removed at most five tasks from OPT , one can show easily that at the end we select $\lceil k/5 \rceil$ tasks, assuming that $|\text{OPT}| \geq k$. Our running time is $k^{O(k)} n^{O(1)}$ since in each of the k iterations there are $O(k^2)$ options for our guesses.

► **Theorem 24.** *There is an FPT-5-approximation algorithm for UFT with a running time of $k^{O(k)} n^{O(1)}$.*

References

- 1 Anna Adamaszek, Parinya Chalermsook, Alina Ene, and Andreas Wiese. Submodular unsplittable flow on trees. *Math. Program.*, 172(1-2):565–589, 2018. doi:10.1007/s10107-017-1218-4.
- 2 Marek Adamczyk, Jarosław Byrka, Jan Marcinkowski, Syed Mohammad Meesum, and Michał Włodarczyk. Constant factor FPT approximation for capacitated k-median. *CoRR*, abs/1809.05791, 2018. arXiv:1809.05791.
- 3 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4):FOCS17–97, 2019.
- 4 E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8, 1987.
- 5 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.
- 6 Reuven Bar-Yehuda, Michael Beder, Yuval Cohen, and Dror Rawitz. Resource allocation in bounded degree trees. *Algorithmica*, 54(1):89–106, 2009.
- 7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.
- 8 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, 2014.

- 9 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 Theory of computing (STOC 1998)*, pages 114–123, 1998.
- 10 C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.
- 11 C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.
- 12 Bo Chen, Refael Hassin, and Michal Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34(5):501–507, 2002.
- 13 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In *International Symposium on Parameterized and Exact Computation*, pages 110–122. Springer, 2013.
- 14 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.
- 15 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.42.
- 16 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.41.
- 17 Andreas Emil Feldmann, CS Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.
- 18 Zachary Friggstad and Zhihan Gao. On Linear Programming Relaxations for Unsplittable Flow in Trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 265–283, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.265.
- 19 N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997. doi:10.1007/BF02523685.
- 20 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*, pages 607–619, 2018. doi:10.1145/3188745.3188894.
- 21 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999.
- 22 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 585–594, 2003.
- 23 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *International Colloquium on Automata, Languages, and Programming*, pages 77–88. Springer, 2011.
- 24 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.
- 25 Andreas Wiese. A $(1 + \epsilon)$ -approximation for unsplittable flow on a path in fixed-parameter running time. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 67:1–67:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.

A Simple Algorithm for Graph Reconstruction

Claire Mathieu ✉🏠

CNRS, IRIF, Université de Paris, France

Hang Zhou ✉🏠

École Polytechnique, Institut Polytechnique de Paris, France

Abstract

How efficiently can we find an unknown graph using distance queries between its vertices? We assume that the unknown graph is connected, unweighted, and has bounded degree. The goal is to find every edge in the graph. This problem admits a reconstruction algorithm based on multi-phase Voronoi-cell decomposition and using $\tilde{O}(n^{3/2})$ distance queries [27].

In our work, we analyze a simple reconstruction algorithm. We show that, on random Δ -regular graphs, our algorithm uses $\tilde{O}(n)$ distance queries. As by-products, we can reconstruct those graphs using $O(\log^2 n)$ queries to an all-distances oracle or $\tilde{O}(n)$ queries to a betweenness oracle, and we bound the metric dimension of those graphs by $\log^2 n$.

Our reconstruction algorithm has a very simple structure, and is highly parallelizable. On general graphs of bounded degree, our reconstruction algorithm has subquadratic query complexity.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Random network models; Networks → Network algorithms

Keywords and phrases reconstruction, network topology, random regular graphs, metric dimension

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.68

Related Version Full Version: <http://www.normalesup.org/~zhou>

Funding This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

Acknowledgements We want to thank the anonymous reviewers for their valuable comments.

1 Introduction

Discovering the topology of the Internet is a crucial step for building accurate network models and designing efficient algorithms for Internet applications. The topology of Internet networks is typically investigated at the router level, using `traceroute`. It is a common and reasonably accurate assumption that `traceroute` generates paths that are shortest in the network. Unfortunately, sometimes routers block `traceroute` requests due to privacy and security concerns. As a consequence, the inference of the network topology is rather based on the end-to-end delay information on those requests, which is roughly proportional to the shortest-path distances in the network.

In the *graph reconstruction* problem, we are given the vertex set V of a hidden connected, undirected, and unweighted graph and have access to information about the topology of the graph via an oracle, and the goal is to find every edge in E . Henceforth, unless explicitly mentioned, all graphs studied are assumed to be connected. This assumption is standard and shared by almost all references on the subject, e.g., [7, 14, 27, 39, 41]. The efficiency of an algorithm is measured by the *query complexity*, i.e., the number of queries to the oracle. Motivated by `traceroute`, the literature has explored several types of query oracles.

- One type consists of *all-shortest-paths* and *all-distances* queries, when querying a vertex yields either shortest paths from that vertex to all other vertices [7, 41] or distances from that vertex to all other vertices [14]. The latter, of course, is less informative.
- A more refined type of query oracles, suggested in [7, 14], consists of *shortest-path* and *distance* queries, when querying a pair of vertices yields either a shortest path or the distance between them [27, 38, 39]. Again, the latter is less informative.



© Claire Mathieu and Hang Zhou;
licensed under Creative Commons License CC-BY 4.0
29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 68; pp. 68:1–68:18

Leibniz International Proceedings in Informatics



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

In this work, we focus on the weakest of those four query oracles, that takes as input a pair of vertices a and b and returns the distance $\delta(a, b)$ between them. Reyzin and Srivastava [38] showed that graph reconstruction requires $\Omega(n^2)$ distance queries on general graphs, so we focus on the bounded degree case. For graphs of bounded degree, Kannan, Mathieu, and Zhou [27] gave a reconstruction algorithm based on multi-phase Voronoi-cell decomposition and using $\tilde{O}(n^{3/2})$ distance queries, and raised an open question of whether $\tilde{O}(n)$ is achievable.¹

We provide a partial answer to that open question by analyzing a simple reconstruction algorithm (Algorithm 1). We show that, on (uniformly) random Δ -regular graphs, where every vertex has the same degree Δ , our reconstruction algorithm uses $\tilde{O}(n)$ distance queries (Theorem 1). As by-products, we can reconstruct those graphs using $O(\log^2 n)$ queries to an all-distances oracle (Corollary 2) or using $\tilde{O}(n)$ queries to a betweenness oracle (Corollary 3), and we bound the metric dimension of those graphs by at most $\log^2 n$ (Corollary 5).

Our analysis exploits the *locally tree-like* property of random Δ -regular graphs, meaning that these graphs contain a small number of short cycles. Our method might be applicable to other locally tree-like graphs, such as Erdős-Rényi random graphs and *scale-free* graphs. In particular, many real world networks, such as Internet networks, social networks, and peer-to-peer networks, are believed to have scale-free properties [6, 25, 34]. We defer the reconstruction of those networks for future work.

Our reconstruction algorithm has a very simple structure, and is highly parallelizable (Corollary 4). On general graphs of bounded degree, the same reconstruction algorithm has subquadratic query complexity (Theorem 6).

1.1 Related Work

The problem of reconstructing a graph using queries that reveal partial information has been extensively studied in different contexts and has many applications.

Reconstruction of Random Graphs

The gist of our paper deals with random graphs. The graph reconstruction problem has already attracted much interest in the setting of random graphs. On Erdős-Rényi random graphs, Erlebach, Hall, and Mihalák [15] studied the approximate network reconstruction using all-shortest-paths queries; Anandkumar, Hassidim, and Kelner [4] used end-to-end measurements between a subset of vertices to approximate the network structure. Experimental results to reconstruct random graphs using shortest-path queries were given in [8, 20].

On random Δ -regular graphs, Achlioptas et al. [2] studied the bias of `traceroute` sampling in the context of the network reconstruction. They showed that the structure revealed by `traceroute` sampling on random Δ -regular graphs admits a power-law degree distribution [2], a common phenomenon as in Erdős-Rényi random graphs [31] and Internet networks [16].

Metric Dimension and Related Problems

Our work yields an upper bound on the *metric dimension* of random Δ -regular graphs. The metric dimension problem was first introduced by Slater [42] and Harary and Melter [21], see also [5, 13, 12, 23, 29, 36, 37, 40]. The metric dimension of a graph is the cardinality of

¹ The notation $\tilde{O}(f(n))$ stands for $O(f(n) \cdot \text{polylog } f(n))$.

a smallest subset S of vertices such that every vertex in the graph has a unique vector of distances to the vertices in S . On regular graphs, the metric dimension problem was studied in special cases [12, 24]. In Erdős-Rényi random graphs, the metric dimension problem was studied by Bollobás, Mitsche, and Prałat [11]. Mitsche and Rué [32] also considered the random forest model.

A related problem is the *identifying code* of a graph [28], which is a smallest subset of vertices such that every vertex of the graph is uniquely determined by its neighbourhood within this subset. The identifying code problem was studied on random Δ -regular graphs [17] and on Erdős-Rényi random graphs [19]. Other related problems received attentions on random graphs as well, such as the *sequential metric dimension* [35] and the *seeded graph matching* [33].

Betweenness Oracle

There exists an oracle that is even weaker than the distance oracle: the *betweenness* oracle [1], which receives three vertices u , v , and w and returns whether w lies on a shortest path between u and v . Our work yields a reconstruction algorithm using $\tilde{O}(n)$ betweenness queries on random Δ -regular graphs. For graphs of bounded degree, Abrahamsen et al. [1] generalized the $\tilde{O}(n^{3/2})$ result in the distance oracle model from [27] to the betweenness oracle model.

Tree Reconstruction and Parallel Setting

Our paper focuses on the distance oracle and bounded degree, and considers the parallel setting. All of those aspects were previously raised in the special case of the *tree reconstruction*. Indeed, motivated by the reconstruction of a phylogenetic tree in evolutionary biology, the tree reconstruction problem using a distance oracle is well-studied [22, 30, 43], in particular assuming bounded degree [22]. Afshar et al. [3] studied the tree reconstruction in the parallel setting, analyzing both the *round complexity* and the *query complexity* in the relative distance query model [26].

1.2 Our Results

Our reconstruction algorithm, called SIMPLE, is given in Algorithm 1. It takes as input the vertex set V of size n and an integer parameter $s \in [1, n]$.

■ **Algorithm 1** SIMPLE (V, s).

-
- 1: $S \leftarrow$ sample of s vertices selected uniformly and independently at random from V
 - 2: **for** $u \in S$ and $v \in V$ **do** QUERY(u, v)
 - 3: $\hat{E} \leftarrow$ set of vertex pairs $\{a, b\} \subseteq V$ such that, for all $u \in S$, $|\delta(u, a) - \delta(u, b)| \leq 1$
 - 4: **for** $\{a, b\} \in \hat{E}$ **do** QUERY(a, b)
 - 5: **return** set of vertex pairs $\{a, b\} \in \hat{E}$ such that $\delta(a, b) = 1$
-

Intuitively, the set \hat{E} constructed in SIMPLE consists of all vertex pairs $\{a, b\} \subseteq V$ that *might* be an edge in E . In order to obtain the edge set E , it suffices to query uniquely the vertex pairs in \hat{E} . We remark that SIMPLE correctly reconstructs the graph for any parameter $s \in [1, n]$, and that choosing an appropriate s only affects the query complexity, see Lemma 9.

1.2.1 Random Regular Graphs

Our first main result shows that SIMPLE (Algorithm 1) uses $\tilde{O}(n)$ distance queries on random Δ -regular graphs for an appropriately chosen s (Theorem 1). The analysis exploits the *locally tree-like* property of random Δ -regular graphs. The proof of Theorem 1 consists of several technical novelties, based on a new concept of *interesting vertices* (Definition 14). See Section 3.

► **Theorem 1.** *Consider a uniformly random Δ -regular graph with $\Delta = O(1)$. Let $s = \log^2 n$. In the distance query model, SIMPLE (Algorithm 1) is a reconstruction algorithm using $\tilde{O}(n)$ queries in expectation.*

We extend SIMPLE and its analysis to reconstruct random Δ -regular graphs in the all-distances query model (Corollary 2), in the betweenness query model (Corollary 3), as well as in the parallel setting (Corollary 4). These extensions are based on the observation that the set \hat{E} constructed in SIMPLE equals the edge set E with high probability (Lemma 17),² see Section 4.

► **Corollary 2.** *Consider a uniformly random Δ -regular graph with $\Delta = O(1)$. In the all-distances query model, there is a reconstruction algorithm using $O(\log^2 n)$ queries in expectation.*

► **Corollary 3.** *Consider a uniformly random Δ -regular graph with $\Delta = O(1)$. In the betweenness query model, there is a reconstruction algorithm using $\tilde{O}(n)$ queries in expectation.*

► **Corollary 4.** *Consider a uniformly random Δ -regular graph with $\Delta = O(1)$. In the parallel setting of the distance query model, there is a reconstruction algorithm using $1 + o(1)$ rounds and $\tilde{O}(n)$ queries in expectation.*

We further extend the analysis of SIMPLE to study the metric dimension of random Δ -regular graphs (Corollary 5), by showing (in Lemma 21) that a random subset of $\log^2 n$ vertices is almost surely a *resolving set* (Definition 20) for those graphs, see Section 5.

► **Corollary 5.** *Consider a uniformly random Δ -regular graph with $\Delta = O(1)$. With probability $1 - o(1)$, the metric dimension of the graph is at most $\log^2 n$.*

With extra work, the parameter $s = \log^2 n$ in Theorem 1 can be reduced to $\log n \cdot (\log \log n)^{2+\epsilon}$, for any $\epsilon > 0$, see the full version of the paper. As a consequence, the query complexity in the all-distances query model (Corollary 2) and the upper bound on the metric dimension (Corollary 5) can both be improved to $O(\log n \cdot (\log \log n)^{2+\epsilon})$.

1.2.2 Bounded-Degree Graphs

On general graphs of bounded degree, SIMPLE (Algorithm 1) has subquadratic query complexity and is highly parallelizable (Theorem 6), see Section 6.

► **Theorem 6.** *Consider a general graph of bounded degree $\Delta = O(\text{polylog } n)$. Let $s = n^{2/3}$. In the distance query model, SIMPLE (Algorithm 1) is a reconstruction algorithm using $\tilde{O}(n^{5/3})$ queries in expectation. In addition, SIMPLE can be parallelized using 2 rounds.*

² This property (i.e., $\hat{E} = E$ with high probability) does not hold on general graphs of bounded degree.

We note that the MULTI-PHASE algorithm³ from [27] also reconstructs graphs of bounded degree in the distance query model. How does SIMPLE compare to MULTI-PHASE? In terms of query complexity, on general graphs of bounded degree, SIMPLE uses $\tilde{O}(n^{5/3})$ queries, so is not as good as MULTI-PHASE using $\tilde{O}(n^{3/2})$ queries; on random Δ -regular graphs, SIMPLE is more efficient than MULTI-PHASE: $\tilde{O}(n)$ versus $\tilde{O}(n^{3/2})$. In terms of round complexity, SIMPLE can be parallelized using 2 rounds on general graphs, and even $1 + o(1)$ rounds on random Δ -regular graphs; while MULTI-PHASE requires up to $3 \log n$ rounds due to a multi-phase selection process for centers.⁴ In terms of structure, SIMPLE is much simpler than MULTI-PHASE, which is based on multi-phase Voronoi-cell decomposition.

► **Remark.** In the worst case, the query complexity of SIMPLE is higher than linear. For example, when the graph is a complete binary tree, SIMPLE would require $\Omega(n\sqrt{n})$ queries (the complexity of SIMPLE is minimized when s is roughly \sqrt{n}). Thus the open question from [27] of whether general graphs of bounded degree can be reconstructed using $\tilde{O}(n)$ distance queries remains open and answering it positively would require further algorithmic ideas.

2 Notations and Preliminary Analysis

Let $G = (V, E)$ be a connected, undirected, and unweighted graph, where V is the set of vertices such that $|V| = n$ and E is the set of edges. We say that $\{a, b\} \subseteq V$ is a *vertex pair* if both a and b belong to V such that $a \neq b$. The *distance* between a vertex pair $\{a, b\} \subseteq V$, denoted by $\delta(a, b)$, is the number of edges on a shortest a -to- b path.

► **Definition 7** (Distinguishing). *For a vertex pair $\{a, b\} \subseteq V$, we say that a vertex $u \in V$ distinguishes a and b , or equivalently that u is a distinguisher of $\{a, b\}$, if $|\delta(u, a) - \delta(u, b)| > 1$. Let $D(a, b) \subseteq V$ denote the set of vertices $u \in V$ distinguishing a and b .*

Let $s \in [1, n]$ be an integer parameter. The set S constructed in SIMPLE consists of s vertices selected uniformly and independently at random from V .

The set \hat{E} constructed in SIMPLE consists of the vertex pairs $\{a, b\} \subseteq V$ such that a and b are not distinguished by any vertex in S , i.e., $D(a, b) \cap S = \emptyset$, or equivalently, $|\delta(u, a) - \delta(u, b)| \leq 1$ for all $u \in S$. For any edge $(a, b) \in E$, it is easy to see that $|\delta(u, a) - \delta(u, b)| \leq 1$ for all $u \in V$, which implies that $\{a, b\} \in \hat{E}$. Hence the following inclusion.

► **Fact 8.** $E \subseteq \hat{E}$.

We show that SIMPLE is correct and we give a preliminary analysis on its query complexity as well as on its round complexity, in Lemma 9.

► **Lemma 9.** *The output of SIMPLE (Algorithm 1) equals the edge set E . The number of distance queries in SIMPLE is $n \cdot s + |\hat{E}|$. In addition, SIMPLE can be parallelized using 2 rounds.*

Proof. The output of SIMPLE consists of the vertex pairs $\{a, b\} \in \hat{E}$ such that $\{a, b\}$ is an edge in E . Since $E \subseteq \hat{E}$ (Fact 8), the output of SIMPLE equals the edge set E .

Observe that the distance queries in SIMPLE are performed in two stages. The number of distance queries in the first stage is $|V| \cdot |S| = n \cdot s$. The number of distance queries in

³ Algorithm 3 in [27].

⁴ The number of rounds in MULTI-PHASE is implicit in the proof of Lemma 2.3 from [27].

the second stage is $|\hat{E}|$. Thus the query complexity of SIMPLE is $n \cdot s + |\hat{E}|$. The distance queries in each of the two stages can be performed in parallel, so SIMPLE can be parallelized using 2 rounds. \blacktriangleleft

From Lemma 9, in order to further study the query complexity of SIMPLE, it suffices to analyze $|\hat{E}|$, which equals $|E| + |\hat{E} \setminus E|$ according to Fact 8. Since $|E| \leq \Delta n$ in a graph of bounded degree Δ , our focus in the subsequent analysis is $|\hat{E} \setminus E|$.

► **Lemma 10.** *Let $s = \omega(\log n)$ be an integer parameter. Let B be the set of vertex pairs $\{a, b\} \subseteq V$ such that $\delta(a, b) \geq 2$ and $|D(a, b)| \leq 3n \cdot (\log n)/s$. We have $\mathbb{E}_S[|\hat{E} \setminus E|] \leq |B| + o(1)$.*

Proof. Denote Z as the set $\hat{E} \setminus E$. Observe that $|Z| \leq |B| + |Z \setminus B|$. Since B is independent of the random set S , we have $\mathbb{E}_S[|Z|] \leq |B| + \mathbb{E}_S[|Z \setminus B|]$. It suffices to show that $\mathbb{E}_S[|Z \setminus B|] = o(1)$.

We claim that for any vertex pair $\{a, b\} \subseteq V$ such that $\{a, b\} \notin B$, the probability that $\{a, b\} \in Z$ is $o(n^{-2})$. To see this, fix a vertex pair $\{a, b\} \notin B$. By definition of B , either $\delta(a, b) = 1$, or $|D(a, b)| > 3n \cdot (\log n)/s$. In the first case, $\{a, b\} \notin Z$ since Z does not contain any edge of E . In the second case, observe that the event $\{a, b\} \in Z$ implies that $\{a, b\} \in \hat{E}$, hence $D(a, b) \cap S = \emptyset$. Therefore,

$$\begin{aligned} & \mathbb{P}_S[\{a, b\} \in Z \mid \{a, b\} \notin B] \\ & \leq \mathbb{P}_S[D(a, b) \cap S = \emptyset \mid \{a, b\} \notin B] \\ & < \left(1 - \frac{3n \cdot (\log n)/s}{n}\right)^s \\ & = o(n^{-2}), \end{aligned}$$

where the second inequality follows since $|D(a, b)| > 3n \cdot (\log n)/s$ and the set S consists of s vertices selected uniformly and independently at random, and the last step follows since $s = \omega(\log n)$.

There are at most $n(n-1)/2$ vertex pairs $\{a, b\} \notin B$. By the linearity of expectation, the expected number of vertex pairs $\{a, b\} \notin B$ such that $\{a, b\} \in Z$ is at most $o(n^{-2}) \cdot n(n-1)/2 = o(1)$, so $\mathbb{E}_S[|Z \setminus B|] = o(1)$. Therefore, $\mathbb{E}_S[|Z|] \leq |B| + \mathbb{E}_S[|Z \setminus B|] = |B| + o(1)$. \blacktriangleleft

3 Reconstruction of Random Regular Graphs (Proof of Theorem 1)

In this section, we analyze SIMPLE (Algorithm 1) on random Δ -regular graphs in the distance query model. We assume that $\Delta \geq 2$ and that Δn is even since otherwise those graphs do not exist.

We bound the expectation of $|\hat{E} \setminus E|$ on random Δ -regular graphs, in Lemma 11.

► **Lemma 11.** *Let G be a uniformly random Δ -regular graph with $\Delta = O(1)$. Let $s = \log^2 n$. Let $S \subseteq V$ be a set of s vertices selected uniformly and independently at random from V . We have $\mathbb{E}_{G,S}[|\hat{E} \setminus E|] = o(1)$.*

Proof of Theorem 1 using Lemma 11. By Lemma 9, SIMPLE is a reconstruction algorithm using $n \cdot s + |\hat{E}| = n \cdot \log^2 n + |\hat{E}|$ distance queries. From Fact 8, $|\hat{E}| = |E| + |\hat{E} \setminus E|$. Since G is Δ -regular, $|E| = \Delta n/2$. By Lemma 11, $\mathbb{E}_{G,S}[|\hat{E} \setminus E|] = o(1)$. Therefore, the expected number of distance queries in SIMPLE is $n \cdot \log^2 n + \Delta n/2 + o(1)$, which is $\tilde{O}(n)$ since $\Delta = O(1)$. \blacktriangleleft

It remains to prove Lemma 11 in the rest of this section.

Configuration Model [9, 44]. We consider a random Δ -regular graph generated according to the *configuration model*. Given a partition of a set of Δn points into n cells v_1, v_2, \dots, v_n of Δ points, a *configuration* is a perfect matching of the points into $\Delta n/2$ pairs. It corresponds to a (not necessarily connected) *multigraph* G' in which the cells are regarded as vertices and the pairs as edges: a pair of points $\{x, y\}$ in the configuration corresponds to an edge (v_i, v_j) of G' where $x \in v_i$ and $y \in v_j$. Since each Δ -regular graph has exactly $(\Delta!)^n$ corresponding configurations, a Δ -regular graph can be generated uniformly at random by rejection sampling: choose a configuration uniformly at random,⁵ and reject the result if the corresponding multigraph G' is not simple or not connected. The configuration model enables us to show properties of a random Δ -regular graph by analyzing a multigraph G' corresponding to a random configuration.

In order to prove Lemma 11, we need the following Structural Lemma.

► **Lemma 12 (Structural Lemma).** *Let $\Delta = O(1)$ be such that $\Delta \geq 3$. Let G' be a multigraph corresponding to a uniformly random configuration. Let $\{v, w\}$ be a vertex pair in G' such that $\delta(v, w) \geq 2$. With probability $1 - o(n^{-2})$, we have $|D(v, w)| > 3n/\log n$.*

We defer the proof of the Structural Lemma for the moment and first show how it implies Lemma 11.⁶

Proof of Lemma 11 using the Structural Lemma (Lemma 12). Let G be a random graph and let S be a random subset of vertices, both defined in the statement of Lemma 11. According to Lemma 10, $\mathbb{E}_{G,S}[\hat{E} \setminus E] \leq \mathbb{E}_G[|B|] + o(1)$. It suffices to prove that $\mathbb{E}_G[|B|] = o(1)$.

First, we consider the case when $\Delta = O(1)$ is such that $\Delta \geq 3$. Our analysis is based on the configuration model. Let G' be a multigraph corresponding to a uniformly random configuration. Let $\mathbb{E}_{G'}[|B|]$ denote the expected size of the set B defined on G' . Since each Δ -regular graph corresponds to the same number of configurations and because the probability spaces of configurations and of Δ -regular graphs, respectively, are uniform, we have $\mathbb{E}_G[|B|] \leq \mathbb{E}_{G'}[|B|]/p$, where p is the probability that G' is both simple and connected. According to [44], when $\Delta \geq 3$, $p \sim e^{(1-\Delta^2)/4}$, which is constant since $\Delta = O(1)$. Thus $\mathbb{E}_G[|B|] = O(\mathbb{E}_{G'}[|B|])$.

In order to bound $\mathbb{E}_{G'}[|B|]$, consider any vertex pair $\{v, w\}$ in G' such that $\delta(v, w) \geq 2$. From Lemma 12, the event $|D(v, w)| \leq 3n/\log n$ occurs with probability $o(n^{-2})$. Equivalently, the event $|D(v, w)| \leq 3n \cdot (\log n)/s$ occurs with probability $o(n^{-2})$, since $s = \log^2 n$. Thus the event $\{v, w\} \in B$ occurs with probability $o(n^{-2})$ according to the definition of B in Lemma 10. There are $n(n-1)/2$ vertex pairs $\{v, w\}$ in G' . By linearity of expectation, $\mathbb{E}_{G'}[|B|]$ is at most $o(n^{-2}) \cdot n(n-1)/2 = o(1)$. Hence $\mathbb{E}_G[|B|] = O(\mathbb{E}_{G'}[|B|]) = o(1)$.

In the special case when $\Delta = 2$, a 2-regular graph G is a ring. Consider any vertex pair $\{v, w\}$ in G such that $\delta(v, w) \geq 2$. It is easy to see that at least $n-4$ vertices u in the ring G are such that $|\delta(u, v) - \delta(u, w)| > 1$, so $|D(v, w)| \geq n-4$ by Definition 7. When n is large enough, $n-4 > 3n/\log n$, so $|D(v, w)| > 3n/\log n$. Equivalently, we have $|D(v, w)| > 3n \cdot (\log n)/s$, since $s = \log^2 n$. Thus $\{v, w\} \notin B$ according to the definition of B in Lemma 10. Therefore, $B = \emptyset$ and $\mathbb{E}_G[|B|] = 0$.

We conclude that $\mathbb{E}_G[|B|] = o(1)$ for any $\Delta = O(1)$. Thus $\mathbb{E}_{G,S}[\hat{E} \setminus E] \leq \mathbb{E}_G[|B|] + o(1) = o(1)$. ◀

⁵ To generate a random configuration, the points in a pair can be chosen sequentially: the first point can be selected using any rule, as long as the second point in that pair is chosen uniformly from the remaining points.

⁶ Note that Lemma 11 is an intermediate step to prove Theorem 1 using Lemma 12. We state Lemma 11 separately since it will be reused in Section 4.

The rest of the section is dedicated to prove the Structural Lemma (Lemma 12).

Let G' be a multigraph corresponding to a uniformly random configuration, and let V be the vertex set of G' . Let $\{v, w\} \subseteq V$ be a vertex pair such that $\delta(v, w) \geq 2$. For a vertex $x \in V$, denote $\ell(x) \in \mathbb{Z}$ as the distance in G' between x and the vertex pair $\{v, w\}$, i.e., $\ell(x) = \min(\delta(x, v), \delta(x, w))$. For any integer $k \geq 0$, denote $U_k \subseteq V$ as the set of vertices $x \in V$ such that $\ell(x) = k$. Denote $U_{\leq k} = \bigcup_{j \leq k} U_j$.

To construct the multigraph G' from a random configuration, we borrow the approach from [10], which proceeds in phases to construct the edges in G' , exploring vertices $x \in V$ in non-decreasing order of $\ell(x)$. We start at the vertices of $U_0 = \{v, w\}$. Initially (i.e., in the 0-th phase), we construct all the edges incident to v or incident to w . Suppose at the beginning of the k -th phase (for $k \in [1, n-1]$), we have constructed all the edges with at least one endpoint belonging to $U_{\leq k-1}$. During the k -th phase, we construct the edges incident to the vertices in U_k one by one, till the degree of all the vertices in U_k reaches Δ . Let G' be the resulting multigraph in the end of the construction.⁷

An edge (a, b) in G' is *indispensable* if it explores either the vertex a or the vertex b for the first time in the edge construction. In the first case, b is the *predecessor* of a ; and in the second case, a is the *predecessor* of b . An edge is *dispensable* if it is not indispensable, in other words, if each of its endpoints either belongs to $\{v, w\}$ or is an endpoint of an edge constructed previously.

► **Fact 13.** *Neither v or w has a predecessor. For any vertex in V , its predecessor, if exists, is unique. If vertex a is the predecessor of vertex b , then $\ell(b) = \ell(a) + 1$.*

We introduce the concept of *interesting vertices*, which is a key idea in the analysis.

► **Definition 14 (Interesting Vertices).** *A vertex $x \in V$ is v -interesting if, for all vertices $z \in V \setminus \{v\}$ with $\delta(v, z) + \delta(z, x) = \delta(v, x)$, the edges incident to z are indispensable. Similarly, a vertex $x \in V$ is w -interesting if, for all vertices $z \in V \setminus \{w\}$ with $\delta(w, z) + \delta(z, x) = \delta(w, x)$, the edges incident to z are indispensable.*

For any finite integer $k \geq 1$, let $I_k(v) \subseteq V$ denote the set of v -interesting vertices $x \in V$ such that $\delta(v, x) = k$, and let $I_k(w) \subseteq V$ denote the set of w -interesting vertices $x \in V$ such that $\delta(w, x) = k$.

We show in Lemma 15 that interesting vertices distinguish the vertex pair $\{v, w\}$, which is a main technical novelty of the section.

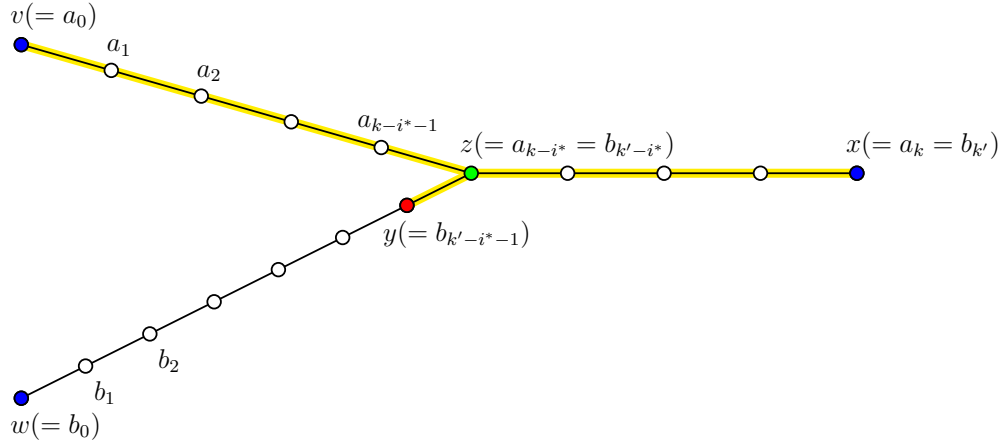
► **Lemma 15.** *For any finite integer $k \geq 1$, we have $I_k(v) \cup I_k(w) \subseteq D(v, w)$.*

Proof. Fix a finite integer $k \geq 1$. From the symmetry of v and w , it suffices to prove $I_k(v) \subseteq D(v, w)$.

Let x be any vertex in $I_k(v)$. By definition, x is v -interesting and $\delta(v, x) = k$. Let $a_0 = v, a_1, \dots, a_k = x$ be any shortest v -to- x path. For any vertex a_i with $i \in [1, k]$, the edges incident to a_i are indispensable according to Definition 14.

We claim that, for any $i \in [1, k]$, a_{i-1} is the predecessor of a_i , and in addition, $\ell(a_i) = i$. The proof is by induction. First, consider the case when $i = 1$. The edge (a_0, a_1) is incident to the vertex a_1 , so is indispensable. Thus either a_0 is the predecessor of a_1 , or a_1 is the

⁷ When a multigraph corresponding to a random configuration is not connected, the resulting G' consists of the union of the components of v and of w , respectively, in that multigraph. It is not necessary to extend G' to the entire multigraph. Indeed, any vertex $x \in V$ outside the union of those two components cannot distinguish v and w (i.e., $x \notin D(v, w)$), thus x is irrelevant to $|D(v, w)|$ in the statement of Lemma 12.



■ **Figure 1** a_0, a_1, \dots, a_k is a shortest v -to- x path, and $b_0, b_1, \dots, b_{k'}$ is a shortest w -to- x path. The vertex z represents the branching point of these two paths. Since the vertex x is v -interesting, the highlighted edges are indispensable.

predecessor of a_0 . Since $a_0 (= v)$ has no predecessor (Fact 13), a_1 cannot be the predecessor of a_0 , so a_0 is the predecessor of a_1 . Again using Fact 13, we have $\ell(a_1) = \ell(a_0) + 1$. Since $\ell(a_0) = \ell(v) = 0$, we have $\ell(a_1) = 1$. Next, consider the case when $i \geq 2$, and assume that the claim holds already for $1, \dots, i-1$. The edge (a_{i-1}, a_i) is incident to the vertex a_i , so is indispensable. Thus either a_{i-1} is the predecessor of a_i , or a_i is the predecessor of a_{i-1} . By induction, a_{i-2} is the predecessor of a_{i-1} . Since the predecessor of a_{i-1} is unique (Fact 13), a_i cannot be the predecessor of a_{i-1} , so a_{i-1} is the predecessor of a_i . Again using Fact 13, we have $\ell(a_i) = \ell(a_{i-1}) + 1$. Since $\ell(a_{i-1}) = i-1$ by induction, we have $\ell(a_i) = i$.

In order to show that $x \in D(v, w)$, we prove in the following that $\delta(w, x) \geq k+2$. Indeed, since $\delta(v, x) = k$, the event $\delta(w, x) \geq k+2$ implies that $x \in D(v, w)$ by Definition 7.⁸

Let $b_0 = w, b_1, \dots, b_{k'} = x$ be any shortest w -to- x path, for some integer k' . See Figure 1. Let $i^* \in [0, k]$ be the largest integer such that $a_{k-j} = b_{k'-j}$ for all $j \in [0, i^*]$. Let z denote the vertex a_{k-i^*} , which equals $b_{k'-i^*}$. If $i^* = k$, the v -to- x path a_0, a_1, \dots, a_k is a subpath of the w -to- x path $b_0, b_1, \dots, b_{k'}$. Since $\delta(w, v) \geq 2$, we have $\delta(w, x) = \delta(w, v) + \delta(v, x) \geq 2 + k$, which implies that $x \in D(v, w)$. From now on, it suffices to consider the case when $i^* < k$.

Let y denote the vertex $b_{k'-i^*-1}$. Since y is on a shortest w -to- x path, we have

$$\delta(w, x) = \delta(w, y) + \delta(y, x) = \delta(w, y) + (i^* + 1) \geq \ell(y) + (i^* + 1), \quad (1)$$

where the inequality follows from the definition of $\ell(y)$. It remains to analyze the value of $\ell(y)$.

The edge (z, y) is incident to the vertex $z (= a_{k-i^*})$, so is indispensable. Thus either y is the predecessor of z , or z is the predecessor of y . From the previous claim, a_{k-i^*-1} is the predecessor of z . Since the predecessor of z is unique (Fact 13) and $y \neq a_{k-i^*-1}$ (by definition of i^*), y cannot be the predecessor of z , so z is the predecessor of y . Again by

⁸ When $\delta(w, x)$ is infinite (i.e., w and x are not connected in G'), it is trivial that $x \in D(v, w)$, since $\delta(v, x)$ is finite. Therefore, it suffices to consider the case when $\delta(w, x)$ is finite in the rest of the proof.

68:10 A Simple Algorithm for Graph Reconstruction

Fact 13, $\ell(y) = \ell(z) + 1$. Since $\ell(z) = \ell(a_{k-i^*}) = k - i^*$ by the previous claim, we have $\ell(y) = k - i^* + 1$. We conclude from Equation (1) that

$$\delta(w, x) \geq (k - i^* + 1) + (i^* + 1) = k + 2,$$

which implies that $x \in D(v, w)$.

We proved that $I_k(v) \subseteq D(v, w)$. Similarly, $I_k(w) \subseteq D(v, w)$. Therefore, $I_k(v) \cup I_k(w) \subseteq D(v, w)$. \blacktriangleleft

A lower bound on the number of interesting vertices is given in Lemma 16, which is another technical novelty. The proof is based on the *locally tree-like* property in random Δ -regular graphs. It exploits the concept of interesting vertices in a non-trivial way and extends a 3-level argument from Bollobás [10] in the context of automorphisms of those graphs to an analysis of $\log \log n$ levels. See the full version of the paper for details.

► **Lemma 16.** *Let $\Delta = O(1)$ be such that $\Delta \geq 3$. Let k be any positive integer such that $k \leq \lceil \log_{\Delta-1}(3n/\log n) \rceil + 2$. With probability $1 - o(n^{-2})$, we have $|I_k(v) \cup I_k(w)| > (\Delta - 2 - o(1))(\Delta - 1)^{k-1}$.*

Proof of the Structural Lemma (Lemma 12). We set $k = \lceil \log_{\Delta-1}(3n/\log n) \rceil + 2$. By Lemma 15, $|D(v, w)| \geq |I_k(v) \cup I_k(w)|$. By Lemma 16, with probability $1 - o(n^{-2})$, we have

$$|I_k(v) \cup I_k(w)| > (\Delta - 2 - o(1))(\Delta - 1)^{k-1} \geq (\Delta - 2 - o(1))(\Delta - 1) \cdot (3n/\log n),$$

where the last inequality follows from the definition of k . Since $\Delta \geq 3$, we have $(\Delta - 2 - o(1))(\Delta - 1) > 1$. Thus with probability $1 - o(n^{-2})$, we have $|I_k(v) \cup I_k(w)| > 3n/\log n$, which implies that $|D(v, w)| > 3n/\log n$. \blacktriangleleft

4 Other Reconstruction Models (Proofs of Corollaries 2–4)

In this section, we study the reconstruction of random Δ -regular graphs in the all-distances query model, in the betweenness query model, as well as in the parallel setting.

By extending the analysis from Section 3, we observe that the set \hat{E} constructed in SIMPLE (Algorithm 1) equals the edge set E with high probability, in Lemma 17.

► **Lemma 17.** *Let G be a uniformly random Δ -regular graph with $\Delta = O(1)$. Let $s = \log^2 n$. Let $S \subseteq V$ be a set of s vertices selected uniformly and independently at random from V . With probability $1 - o(1)$, $|\hat{E}| = \Delta n/2$. In addition, the event $|\hat{E}| = \Delta n/2$ implies $\hat{E} = E$.*

Proof. From Lemma 11, $\mathbb{E}_{G,S}[|\hat{E} \setminus E|] = o(1)$. By Markov's inequality, the event that $|\hat{E} \setminus E| \geq 1$ occurs with probability $o(1)$. Thus with probability $1 - o(1)$, we have $\hat{E} \subseteq E$. On the other hand, $E \subseteq \hat{E}$ by Fact 8. Therefore, the event that $\hat{E} = E$ occurs with probability $1 - o(1)$, and this event occurs if and only if $|\hat{E}| = |E|$. The statement follows since $|E| = \Delta n/2$ in a Δ -regular graph. \blacktriangleleft

4.1 A Modified Algorithm

Lemma 17 enables us to design another reconstruction algorithm in the distance query model, called SIMPLE-MODIFIED, which is a modified version of SIMPLE, see Algorithm 2. SIMPLE-MODIFIED repeatedly computes a set \hat{E} as in SIMPLE, until the size of \hat{E} equals $\Delta n/2$. The parameter s is fixed to $\log^2 n$.

Algorithm 2 SIMPLE-MODIFIED (V).

```

1: repeat
2:    $S \leftarrow$  sample of  $s = \log^2 n$  vertices selected uniformly and independently at random
     from  $V$ 
3:   for  $u \in S$  and  $v \in V$  do QUERY( $u, v$ )
4:    $\hat{E} \leftarrow$  set of vertex pairs  $\{a, b\} \subseteq V$  such that, for all  $u \in S$ ,  $|\delta(u, a) - \delta(u, b)| \leq 1$ 
5: until  $|\hat{E}| = \Delta n/2$ 
6: return  $\hat{E}$ 

```

► **Lemma 18.** *Let G be a uniformly random Δ -regular graph with $\Delta = O(1)$. In the distance query model, SIMPLE-MODIFIED (Algorithm 2) is a reconstruction algorithm, i.e., its output equals the edge set E . The expected number of iterations of the **repeat** loop in SIMPLE-MODIFIED is $1 + o(1)$.*

Proof. Upon termination of the **repeat** loop in SIMPLE-MODIFIED, we have $|\hat{E}| = \Delta n/2$, which implies $\hat{E} = E$ by Lemma 17. Thus the output of SIMPLE-MODIFIED equals the edge set E .

In each iteration of the **repeat** loop, the event that $|\hat{E}| = \Delta n/2$ occurs with probability $1 - o(1)$ by Lemma 17. Thus the expected number of iterations of the **repeat** loop is $1 + o(1)$. ◀

4.2 All-Distances Query Model (Proof of Corollary 2)

By Lemma 18, SIMPLE-MODIFIED is a reconstruction algorithm in the distance query model. We extend SIMPLE-MODIFIED to the all-distances query model.

Observe that in SIMPLE-MODIFIED, the distance queries are performed between each sampled vertex $u \in S$ and all vertices in the graph. This is equivalent to a single query at each sampled vertex $u \in S$ in the all-distances query model. Hence each iteration of the **repeat** loop in SIMPLE-MODIFIED corresponds to $|S| = \log^2 n$ all-distances queries. Again by Lemma 18, the expected number of iterations of the **repeat** loop in SIMPLE-MODIFIED is $1 + o(1)$. Therefore, in the all-distances query model, an algorithm equivalent to SIMPLE-MODIFIED reconstructs the graph using $(1 + o(1)) \cdot \log^2 n = O(\log^2 n)$ all-distances queries in expectation.

4.3 Betweenness Query Model (Proof of Corollary 3)

In the betweenness query model, Abrahamsen et al. [1] showed that $\tilde{O}(\Delta^2 \cdot n)$ betweenness queries suffice to compute the distances from a given vertex to all vertices in the graph (it is implicit in Lemma 16 from [1]), so an all-distances query can be simulated by $\tilde{O}(\Delta^2 \cdot n)$ betweenness queries. As a consequence of Corollary 2, we achieve a reconstruction algorithm using $\tilde{O}(\Delta^2 \cdot n \cdot \log^2 n) = \tilde{O}(n)$ betweenness queries in expectation, since $\Delta = O(1)$.

4.4 Parallel Setting (Proof of Corollary 4)

By Lemma 18, SIMPLE-MODIFIED is a reconstruction algorithm in the distance query model. We analyze SIMPLE-MODIFIED in the parallel setting.

Each iteration of the **repeat** loop consists of $n \cdot \log^2 n$ distance queries, and the distance queries within the same iteration of the **repeat** loop can be performed in parallel. Again by Lemma 18, the expected number of iterations of the **repeat** loop in SIMPLE-MODIFIED is $1 + o(1)$. Thus the expected number of rounds in SIMPLE-MODIFIED is $1 + o(1)$, and the expected number of distance queries in SIMPLE-MODIFIED is $(1 + o(1)) \cdot n \cdot \log^2 n = \tilde{O}(n)$.

5 Metric Dimension (Proof of Corollary 5)

In this section, we study the metric dimension of random Δ -regular graphs. To begin with, we show an elementary structural property of random Δ -regular graphs, in Lemma 19, based on a classical result on those graphs.

► **Lemma 19.** *Let $G = (V, E)$ be a uniformly random Δ -regular graph with $\Delta = O(1)$. With probability $1 - o(1)$, for any edge (a, b) of the graph G , there exists a vertex $c \in V \setminus \{a, b\}$ that is adjacent to b but is not adjacent to a .*

Proof. First, consider the case when $\Delta = 2$. A 2-regular graph is a ring. Let (a, b) be any edge of the graph. The vertex b has two neighbors, the vertex a and another vertex, let it be c . We have $c \in V \setminus \{a, b\}$ and c is not adjacent to a (as soon as $n > 3$). The statement of the lemma follows.

Next, consider the case when $\Delta = O(1)$ is such that $\Delta \geq 3$. Let \mathcal{E} denote the event that, for any edge (a, b) of G , there do not exist two vertices c_1 and c_2 in G , such that all of the 4 edges $(a, c_1), (a, c_2), (b, c_1), (b, c_2)$ belong to G . We show that \mathcal{E} occurs with probability $1 - o(1)$. Indeed, if for some edge (a, b) of G , there exist two vertices c_1 and c_2 such that $(a, c_1), (a, c_2), (b, c_1), (b, c_2)$ are edges of G , then the induced subgraph on $\{a, b, c_1, c_2\}$ consists of at least 5 edges. A classical result on random Δ -regular graphs shows that, for any constant integer k , the probability that there exists an induced subgraph of k vertices with at least $k + 1$ edges is $o(1)$, see, e.g., Lemma 11.12 in [18]. Therefore, \mathcal{E} occurs with probability $1 - o(1)$.

We condition on the occurrence of \mathcal{E} . For any edge (a, b) of G , let $N(a)$ be the set of $\Delta - 1$ neighbors of a that are different from b , and let $N(b)$ be the set of $\Delta - 1$ neighbors of b that are different from a . Since $\Delta \geq 3$, we have $|N(a)| = |N(b)| \geq 2$. The event \mathcal{E} implies that $N(a) \neq N(b)$, so there exists a vertex $c \in N(b) \setminus N(a)$. By definition, c is adjacent to b but is not adjacent to a , and $c \in V \setminus \{a, b\}$. Since \mathcal{E} occurs with probability $1 - o(1)$, we conclude that, with probability $1 - o(1)$, for any edge (a, b) of the graph G , there exists a vertex $c \in V \setminus \{a, b\}$ that is adjacent to b but is not adjacent to a . ◀

► **Definition 20** (e.g., [5, 12]). *A subset of vertices $S \subseteq V$ is a resolving set for a graph $G = (V, E)$ if, for any pair of vertices $\{a, b\} \subseteq V$, there is a vertex $u \in S$ such that $\delta(u, a) \neq \delta(u, b)$. The metric dimension of G is the smallest size of a resolving set for G .*

Based on the analysis of SIMPLE from Lemma 17 and the structural property from Lemma 19, we show that, with high probability, a random subset of $\log^2 n$ vertices is a resolving set for a random Δ -regular graph, in Lemma 21.

► **Lemma 21.** *Let $G = (V, E)$ be a uniformly random Δ -regular graph with $\Delta = O(1)$. Let $S \subseteq V$ be a sample of $s = \log^2 n$ vertices selected uniformly and independently at random from V . With probability $1 - o(1)$, the set S is a resolving set for the graph G .*

Proof. Let \mathcal{E}_1 denote the event that, for any edge (a, b) of the graph G , there exists a vertex $c \in V \setminus \{a, b\}$ that is adjacent to b but is not adjacent to a . By Lemma 19, the event \mathcal{E}_1 occurs with probability $1 - o(1)$. Let \mathcal{E}_2 denote the event $\hat{E} = E$. By Lemma 17, the event \mathcal{E}_2 occurs with probability $1 - o(1)$. Thus with probability $1 - o(1)$, both events \mathcal{E}_1 and \mathcal{E}_2 occur simultaneously. We condition on the occurrences of both events \mathcal{E}_1 and \mathcal{E}_2 in the subsequent analysis.

First, consider any vertex pair $\{a, b\} \subseteq V$ such that $\delta(a, b) \geq 2$. The event \mathcal{E}_2 implies that $\{a, b\} \notin \hat{E}$. By definition, there exists some vertex $u \in S$ such that $|\delta(u, a) - \delta(u, b)| \geq 2$, which implies that $\delta(u, a) \neq \delta(u, b)$.

Next, consider any vertex pair $\{a, b\} \subseteq V$ such that $\delta(a, b) = 1$. The event \mathcal{E}_1 implies that there exists a vertex $c \in V \setminus \{a, b\}$ that is adjacent to b but is not adjacent to a . Since $\delta(a, c) \geq 2$, the event \mathcal{E}_2 implies that $\{a, c\} \notin \hat{E}$. By definition, there exists some vertex $u \in S$ such that $|\delta(u, a) - \delta(u, c)| \geq 2$. Using an elementary inequality of $|x - y| + |y - z| \geq |x - z|$ for any three real numbers x , y , and z , we have

$$\begin{aligned} |\delta(u, a) - \delta(u, b)| &\geq |\delta(u, a) - \delta(u, c)| - |\delta(u, b) - \delta(u, c)| \\ &\geq |\delta(u, a) - \delta(u, c)| - \delta(b, c) && \text{(by the triangle inequality)} \\ &\geq 2 - \delta(b, c) && \text{(by the definition of } u\text{)} \\ &\geq 1 && \text{(since } (b, c) \text{ is an edge in } G\text{).} \end{aligned}$$

Thus $\delta(u, a) \neq \delta(u, b)$.

Therefore, conditioned on the occurrences of both events \mathcal{E}_1 and \mathcal{E}_2 , for any vertex pair $\{a, b\} \subseteq V$, there exists a vertex $u \in S$ such that $\delta(u, a) \neq \delta(u, b)$.

We conclude that, with probability $1 - o(1)$, the set S is a resolving set for G . ◀

From Lemma 21, with probability $1 - o(1)$, the metric dimension of a random Δ -regular graph is at most $\log^2 n$. This completes the proof of Corollary 5.

6 Reconstruction of Bounded-Degree Graphs (Proof of Theorem 6)

In this section, we analyze SIMPLE (Algorithm 1) on general graphs of bounded degree in the distance query model. Recall that a set B of vertex pairs $\{a, b\} \subseteq V$ is defined in Lemma 10. For every vertex $a \in V$, we define the set of vertices $B(a) \subseteq V$ as

$$B(a) = \{b \in V \mid \{a, b\} \in B\}.$$

Intuitively, $B(a)$ consists of the vertices $b \in V$ that has few distinguishers with a . We bound the size of the set $B(a)$ for any vertex a , in Lemma 22.

► **Lemma 22.** *Let G be a general graph of bounded degree Δ . For any vertex $a \in V$, $|B(a)| \leq 9\Delta^3 \cdot n^2 \cdot (\log^2 n)/s^2$.*

We defer the proof of Lemma 22 for the moment and first show how it implies Theorem 6.

Proof of Theorem 6 using Lemma 22. By Lemma 9, SIMPLE is a reconstruction algorithm using $n \cdot s + |\hat{E}|$ distance queries, and in addition, SIMPLE can be parallelized using 2 rounds. It remains to further analyze the query complexity.

From Fact 8, $|\hat{E}| = |E| + |\hat{E} \setminus E|$. Since the graph has bounded degree Δ , $|E| \leq \Delta n$. From Lemma 10, $\mathbb{E}_S[|\hat{E} \setminus E|] \leq |B| + o(1)$. Therefore, the expected number of distance queries in SIMPLE is at most $n \cdot s + \Delta n + |B| + o(1)$. It suffices to analyze $|B|$.

Observe that $|B| \leq \sum_{a \in V} |B(a)|$ by definition of $\{B(a)\}_{a \in V}$. From Lemma 22, $|B(a)| \leq 9\Delta^3 \cdot n^2 \cdot (\log^2 n)/s^2$, for any vertex $a \in V$. Hence $|B| \leq (9\Delta^3 \cdot n^2 \cdot (\log^2 n)/s^2) \cdot n$. Thus the expected number of distance queries in SIMPLE is at most $n \cdot s + \Delta n + (9\Delta^3 \cdot n^2 \cdot (\log^2 n)/s^2) \cdot n + o(1)$, which is $\tilde{O}(n^{5/3})$ since $s = n^{2/3}$ and $\Delta = O(\text{polylog } n)$. ◀

The rest of the section is dedicated to prove Lemma 22.

Let a be any vertex in V . Let T be an (arbitrary) shortest-path tree rooted at a and spanning all vertices in V . For any vertex $b \in V$, let *the shortest a -to- b path* denote the path between a and b in the tree T . To simplify the presentation, we assume that, for any

$b \in B(a)$, $\delta(a, b)$ is even, so that the *midpoint vertex* of the shortest a -to- b path is uniquely defined. We extend our analysis to the general setting in the end of the section.

For any vertex $m \in V$, define the set $B(a, m) \subseteq B(a)$ as

$$B(a, m) = \{b \in B(a) \mid \text{the midpoint vertex of the shortest } a\text{-to-}b \text{ path is } m\}.$$

Define the set $M(a) \subseteq V$ as

$$M(a) = \{m \in V \mid B(a, m) \neq \emptyset\}.$$

In other words, $M(a)$ consists of the vertices $m \in V$ that is the midpoint vertex of the shortest a -to- b path for some $b \in B(a)$. From the construction, we have

$$B(a) = \bigcup_{m \in M(a)} B(a, m). \quad (2)$$

In order to bound the size of $B(a)$, first we bound the size of $B(a, m)$ for any midpoint $m \in M(a)$, in Lemma 23, and then we bound the number of distinct midpoints, in Lemma 24.

► **Lemma 23.** *For any $m \in M(a)$, $|B(a, m)| \leq 3\Delta \cdot n \cdot (\log n)/s$.*

Proof. For any $b \in B(a, m)$, the vertex m is the midpoint vertex of the shortest a -to- b path by definition. From the assumption, $\delta(a, b)$ is even for any $b \in B(a, m)$, so there exists for some positive integer ℓ , such that $\delta(m, a) = \ell$ and $\delta(m, b) = \ell$ for any $b \in B(a, m)$.

For every neighbor m' of m such that $\delta(a, m') = \delta(a, m) + 1$, define a set $Y(m') \subseteq B(a, m)$ that consists of the vertices $b \in B(a, m)$ such that m' is on the shortest a -to- b path. Let \hat{m} be a neighbor of m such that $\delta(a, \hat{m}) = \delta(a, m) + 1$ and that $|Y(\hat{m})|$ is maximized, see Figure 2. Since the graph has bounded degree Δ , we have $|B(a, m)| \leq \Delta \cdot |Y(\hat{m})|$. It suffices to bound $|Y(\hat{m})|$.

The main observation is that any vertex of $Y(\hat{m})$ distinguishes a and any other vertex of $Y(\hat{m})$. To see this, let b_0 be any vertex in $Y(\hat{m})$. By definition, $\delta(a, \hat{m}) = \delta(a, m) + 1 = \ell + 1$. Since \hat{m} is on the shortest a -to- b_0 path, we have $\delta(\hat{m}, b_0) = \delta(a, b_0) - \delta(a, \hat{m}) = \ell - 1$, thus $\delta(\hat{m}, b_0) = \delta(\hat{m}, a) - 2$. For any vertex $b_1 \in Y(\hat{m})$, from the triangle inequalities on δ , we have

$$\delta(b_1, b_0) \leq \delta(b_1, \hat{m}) + \delta(\hat{m}, b_0) = \delta(b_1, \hat{m}) + \delta(\hat{m}, a) - 2 = \delta(b_1, a) - 2.$$

According to Definition 7, the vertex b_1 distinguishes a and b_0 , and equivalently, $b_1 \in D(a, b_0)$. Thus we have $Y(\hat{m}) \subseteq D(a, b_0)$, hence $|Y(\hat{m})| \leq |D(a, b_0)| \leq 3n \cdot (\log n)/s$ using the fact that $b_0 \in Y(\hat{m}) \subseteq B(a)$ and the definition of B in Lemma 10.

We conclude that $|B(a, m)| \leq \Delta \cdot |Y(\hat{m})| \leq 3\Delta \cdot n \cdot (\log n)/s$. ◀

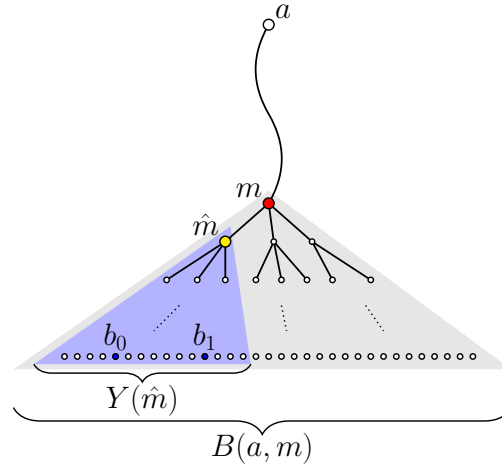
► **Lemma 24.** $|M(a)| \leq 3\Delta \cdot n \cdot (\log n)/s$.

Proof. For each vertex $m \in M(a)$, denote x_m as the second-to-last vertex on the shortest a -to- m path. Denote $X(a) \subseteq V$ as the set of vertices x_m for all $m \in M(a)$. See Figure 3. Since G has bounded degree Δ , we have $|M(a)| \leq \Delta \cdot |X(a)|$. It suffices to bound $|X(a)|$.

Let b^* be a vertex in $B(a)$ such that $\delta(a, b^*)$ is maximized. From the assumption, $\delta(a, b^*)$ is even, so we denote $\delta(a, b^*) = 2\ell$ for some positive integer ℓ .

The main observation is that any vertex of $X(a)$ distinguishes a and b^* . To see this, let x be any vertex in $X(a)$. Let m be any vertex in $M(a)$ such that x is the second-to-last vertex on the shortest a -to- m path.⁹ We have $\delta(a, m) \leq \ell$ and $\delta(a, x) = \delta(a, m) - 1 \leq \ell - 1$. By the

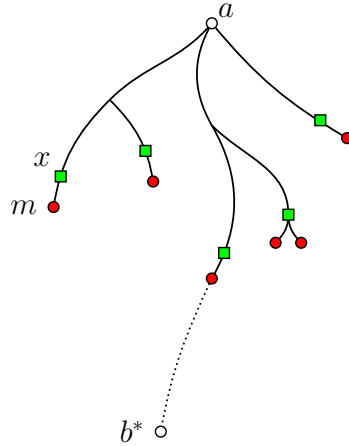
⁹ Such a vertex m exists according to the construction of $X(a)$.



■ **Figure 2** The vertex m is the midpoint of the shortest path between a and any vertex in $B(a, m)$. The vertex \hat{m} is a well-chosen neighbor of m . Consider any vertex $b_0 \in Y(\hat{m})$. We can show that any vertex $b_1 \in Y(\hat{m})$ distinguishes a and b_0 .

triangle inequality on the distances, $\delta(b^*, x) \geq \delta(a, b^*) - \delta(a, x) \geq 2\ell - (\ell - 1) = \ell + 1$. Thus $\delta(b^*, x) - \delta(a, x) \geq 2$. According to Definition 7, the vertex x distinguishes a and b^* , and equivalently, $x \in D(a, b^*)$. Thus $X(a) \subseteq D(a, b^*)$, hence $|X(a)| \leq |D(a, b^*)| \leq 3n \cdot (\log n)/s$ using the fact that $b^* \in B(a)$ and the definition of B in Lemma 10.

We conclude that $|M(a)| \leq \Delta \cdot |X(a)| \leq 3\Delta \cdot n \cdot (\log n)/s$. ◀



■ **Figure 3** Solid circular nodes represent the vertices $m \in M(a)$. Solid curves represent the shortest a -to- m paths. Solid square nodes represent the vertices in $X(a)$. Denote b^* as a vertex in $B(a)$ that is farthest from a . We can show that any vertex $x \in X(a)$ distinguishes a and b^* .

From Equation (2), $|B(a)| \leq \sum_{m \in M(a)} |B(a, m)|$. From Lemma 23, $|B(a, m)| \leq 3\Delta \cdot n \cdot (\log n)/s$ for every $m \in M(a)$. From Lemma 24, $|M(a)| \leq 3\Delta \cdot n \cdot (\log n)/s$. Therefore, $|B(a)| \leq 9\Delta^2 \cdot n^2 \cdot (\log^2 n)/s^2$.

Finally, consider the general setting in which $\delta(a, b)$ is not necessarily even for any $b \in B(a)$. For a vertex m on the shortest a -to- b path, we say that m is the *midpoint vertex* of that path if $\delta(a, m) = \lfloor \delta(a, b)/2 \rfloor$. The definitions of $B(a, m)$ and $M(a)$ remain the same.

Lemma 24 holds in the same way. In Lemma 23, the upper bound of $|B(a, m)|$ is replaced by $3\Delta^2 \cdot n \cdot (\log n)/s$. Indeed, to extend the proof of Lemma 23, instead of considering vertex m' (resp., vertex \hat{m}) that is a neighbor of m , we consider m' (resp., \hat{m}) that is at distance 2 from m . We have $|B(a, m)| \leq \Delta^2 \cdot |Y(\hat{m})|$. The bound $|Y(\hat{m})| \leq 3n \cdot (\log n)/s$ remains the same, so we have $|B(a, m)| \leq 3\Delta^2 \cdot n \cdot (\log n)/s$. Hence $|B(a)| \leq 9\Delta^3 \cdot n^2 \cdot (\log^2 n)/s^2$.

We complete the proof of Lemma 22. Therefore, we obtain Theorem 6.

References

- 1 Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph Reconstruction with a Betweenness Oracle. In *Symposium on Theoretical Aspects of Computer Science*, pages 5:1–5:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
- 2 Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. *Journal of the ACM*, 56(4):21:1–21:28, 2009.
- 3 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing biological and digital phylogenetic trees in parallel. In *European Symposium on Algorithms*, volume 173, pages 3:1–3:24. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.
- 4 Animashree Anandkumar, Avinatan Hassidim, and Jonathan Kelner. Topology discovery of sparse random graphs with few participants. *Random Structures & Algorithms*, 43(1):16–48, 2013.
- 5 Robert F. Bailey and Peter J. Cameron. Base size, metric dimension and other invariants of groups and graphs. *Bulletin of the London Mathematical Society*, 43(2):209–242, 2011.
- 6 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.
- 7 Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matúš Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006.
- 8 Vincent D. Blondel, Jean-Loup Guillaume, Julien M. Hendrickx, and Raphaël M. Jungers. Distance distribution in random graphs and application to network exploration. *Physical Review E*, 76(6):066101, 2007.
- 9 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4):311–316, 1980.
- 10 Béla Bollobás. Distinguishing vertices of random graphs. *North-Holland Mathematics Studies*, 62:33–49, 1982.
- 11 Béla Bollobás, Dieter Mitsche, and Paweł Prałat. Metric dimension for random graphs. *The Electronic Journal of Combinatorics*, 20(4):P1, 2013.
- 12 Gary Chartrand, Linda Eroh, Mark A. Johnson, and Ortrud R. Oellermann. Resolvability in graphs and the metric dimension of a graph. *Discrete Applied Mathematics*, 105(1-3):99–113, 2000.
- 13 José Cáceres, Carmen Hernando, Mercè Mora, Ignacio M. Pelayo, María L. Puertas, Carlos Seara, and David R. Wood. On the metric dimension of cartesian products of graphs. *SIAM Journal on Discrete Mathematics*, 21(2):423–441, 2007.
- 14 Thomas Erlebach, Alexander Hall, Michael Hoffmann, and Matúš Mihalák. Network discovery and verification with distance queries. *Algorithms and Complexity*, pages 69–80, 2006.
- 15 Thomas Erlebach, Alexander Hall, and Matúš Mihalák. Approximate discovery of random graphs. In *International Symposium on Stochastic Algorithms*, pages 82–92. Springer, 2007.
- 16 Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM*, 29(4):251–262, 1999.
- 17 Florent Foucaud and Guillem Perarnau. Bounds for identifying codes in terms of degree parameters. *Electronic Journal of Combinatorics*, 19(P32), 2012.

- 18 Alan Frieze and Michał Karoński. Introduction to random graphs. <https://www.math.cmu.edu/~af1p/BOOK.pdf>.
- 19 Alan Frieze, Ryan Martin, Julien Moncel, Miklós Ruszinkó, and Cliff Smyth. Codes identifying sets of vertices in random networks. *Discrete Mathematics*, 307(9):1094–1107, 2007.
- 20 Jean-Loup Guillaume and Matthieu Latapy. Complex network metrology. *Complex systems*, 16(1):83, 2005.
- 21 Frank Harary and Robert A. Melter. On the metric dimension of a graph. *Ars Combinatoria*, 2(191-195), 1976.
- 22 Jotun J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.
- 23 Carmen Hernando, Merce Mora, Ignacio M. Pelayo, Carlos Seara, and David R. Wood. Extremal graph theory for metric dimension and diameter. *Electronic Notes in Discrete Mathematics*, 29:339–343, 2007. European Conference on Combinatorics, Graph Theory and Applications.
- 24 Imran Javaid, M. Tariq Rahim, and Kashif Ali. Families of regular graphs with constant metric dimension. *Utilitas mathematica*, 75:21–34, 2008.
- 25 Mihajlo Jovanović, Fred Annexstein, and Kenneth Berman. Modeling peer-to-peer network topologies through small-world models and power laws. In *IX Telecommunications Forum, TELFOR*, pages 1–4. Citeseer, 2001.
- 26 Sampath Kannan, Eugene L. Lawler, and Tandy Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996.
- 27 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Transactions on Algorithms*, 14(4):1–30, 2018.
- 28 Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Transactions on Information Theory*, 44(2):599–611, 1998.
- 29 Samir Khuller, Balaji Raghavachari, and Azriel Rosenfeld. Landmarks in graphs. *Discrete applied mathematics*, 70(3):217–229, 1996.
- 30 Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Symposium on Discrete Algorithms*, pages 444–453. SIAM, 2003.
- 31 Anukool Lakhina, John W. Byers, Mark Crovella, and Peng Xie. Sampling biases in IP topology measurements. In *Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 332–341. IEEE, 2003.
- 32 Dieter Mitsche and Juanjo Rué. On the limiting distribution of the metric dimension for random forests. *European Journal of Combinatorics*, 49:68–89, 2015.
- 33 Elchanan Mossel and Jiaming Xu. Seeded graph matching via large neighborhood statistics. *Random Structures & Algorithms*, 57(3):570–611, 2020.
- 34 Mark E. J. Newman, Duncan J. Watts, and Steven H. Strogatz. Random graph models of social networks. *Proceedings of the national academy of sciences*, 99(suppl 1):2566–2572, 2002.
- 35 Gergely Odor and Patrick Thiran. Sequential metric dimension for random graphs, 2020. [arXiv:1910.10116](https://arxiv.org/abs/1910.10116).
- 36 Ortrud R. Oellermann and Joel Peters-Fransen. The strong metric dimension of graphs and digraphs. *Discrete Applied Mathematics*, 155(3):356–364, 2007.
- 37 Yuniór Ramírez-Cruz, Ortrud R. Oellermann, and Juan A. Rodríguez-Velázquez. The simultaneous metric dimension of graph families. *Discrete Applied Mathematics*, 198:241–250, 2016.
- 38 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory*, pages 285–297. Springer, 2007.
- 39 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theoretical Computer Science*, 859:48–56, 2021.
- 40 András Sebő and Eric Tannier. On metric generators of graphs. *Mathematics of Operations Research*, 29(2):383–393, 2004.

- 41 Sandeep Sen and V. N. Muralidhara. The covert set-cover problem with application to network discovery. In *WALCOM*, pages 228–239. Springer, 2010.
- 42 Peter J. Slater. Leaves of trees. In *Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 549–559, 1975.
- 43 Michael S. Waterman, Temple F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.
- 44 Nicholas C. Wormald. Models of random regular graphs. *London Mathematical Society Lecture Note Series*, pages 239–298, 1999.

Generalized Max-Flows and Min-Cuts in Simplicial Complexes

William Maxwell

Oregon State University, Corvallis, OR, USA

Amir Nayyeri

Oregon State University, Corvallis, OR, USA

Abstract

We consider high dimensional variants of the maximum flow and minimum cut problems in the setting of simplicial complexes and provide both algorithmic and hardness results. By viewing flows and cuts topologically in terms of the simplicial (co)boundary operator we can state these problems as linear programs and show that they are dual to one another. Unlike graphs, complexes with integral capacity constraints may have fractional max-flows. We show that computing a maximum integral flow is NP-hard. Moreover, we give a combinatorial definition of a simplicial cut that seems more natural in the context of optimization problems and show that computing such a cut is NP-hard. However, we provide conditions on the simplicial complex for when the cut found by the linear program is a combinatorial cut. For d -dimensional simplicial complexes embedded into \mathbb{R}^{d+1} we provide algorithms operating on the dual graph: computing a maximum flow is dual to computing a shortest path and computing a minimum cut is dual to computing a minimum cost circulation. Finally, we investigate the Ford-Fulkerson algorithm on simplicial complexes, prove its correctness, and provide a heuristic which guarantees it to halt.

2012 ACM Subject Classification Mathematics of computing → Algebraic topology

Keywords and phrases Max-flow min-cut, simplicial complexes, algebraic topology

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.69

Related Version *Full Version*: <https://arxiv.org/abs/2106.14116>

Funding This research was supported in part by NSF grants CCF-1941086, CCF-1816442, and CCF-1617951.

1 Introduction

Computing flows and cuts are fundamental algorithmic problems in graphs, which are one dimensional simplicial complexes. In this paper, we explore generalizations of these algorithmic problems in higher dimensional simplicial complexes. In the case of graphs if an edge $e = (u, v)$ is assigned k units of flow we think of the edge as sending k units of flow from u to v . In two dimensions a flow is an assignment of real valued numbers to the triangles of the simplicial complex. A triangle assigned a value k sends k units of flow around its boundary. The difference in interpretation comes from the fact that the boundary of an edge is disconnected while the boundary of a triangle is connected.

Flows and cuts in simplicial complexes have natural algebraic definitions arising from the theory of simplicial (co)homology. A flow is an element of the kernel of the simplicial boundary operator, and a cut is an element of the image of the simplicial coboundary operator. These subspaces serve as generalizations of the cycle and cut spaces of a graph. This generalization has been studied by Duval, Klivans, and Martin in the setting of CW complexes [7]. We formulate the algorithmic problems of computing max-flows and min-cuts algebraically. By forgetting about the underlying graph structure and focusing on the (co)boundary operators, we obtain methods that naturally generalize to high dimensions.



© William Maxwell and Amir Nayyeri;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 69; pp. 69:1–69:16

Leibniz International Proceedings in Informatics



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

In a graph an st -flow is an assignment of real values to the edges satisfying the conservation of flow constraints: the net flow out of any vertex other than s and t is zero, and, thus, the net flow that leaves s is equal to the net flow that enters t . Therefore, each st -flow can be viewed as a circulation in another graph with an extra edge that connects t to s . Circulations are elements of the cycle space of the graph with coefficients taken over \mathbb{R} . In a d -dimensional simplicial complex \mathcal{K} the d -dimensional cycles are the formal sums (over \mathbb{R}) of d -dimensional simplices whose boundary is zero. Because there are no $(d+1)$ -simplices flows are the elements of the d th homology group $H_d(\mathcal{K}, \mathbb{R})$. The maximum flow problem in a simplicial complex asks to find an optimal element of $H_d(\mathcal{K}, \mathbb{R})$ subject to capacity constraints.

The max-flow min-cut theorem states that in a graph the value of a maximum st -flow is equal to the value of a minimum st -cut. This result is a special case of linear programming duality. By rewriting the linear program in terms of the (co)boundary operator we obtain a similar result for simplicial complexes. The question of whether or not a similar max-flow min-cut theorem holds for simplicial complexes was asked, and left open, in a paper by Latorre [19]. We give a positive answer to this question, but with a caveat. When viewing flows and cuts from a topological point of view their linear programs are dual to one another. However, we also provide a combinatorial definition of a cut which feels more natural for a minimization problem. Topological and combinatorial cuts are equivalent for graphs, but they become different in dimensions $d > 1$. Flows in higher dimension, are dual to topological cuts, but not combinatorial cuts in general. From a computational complexity viewpoint the two notions of cuts are very different. We show that computing a minimum topological cut can be solved via linear programming, but that computing a minimum combinatorial cut is NP-hard.

A closely related problem is the problem of computing a max-flow in a graph which admits an embedding into some topological space. The most well-studied cases are planar graphs and the more general case when the graph embeds into a surface [2, 3, 4, 10, 12, 13, 14, 17, 18, 21, 22]. Max-flows and min-cuts are computationally easier to solve in surface embedded graphs, especially planar graphs. We consider this problem generalized to simplicial complexes. Planar graphs are 1-dimensional complexes embedded in \mathbb{R}^2 , in Section 5 we consider the special case when a d -dimensional simplicial complex admits an embedding into \mathbb{R}^{d+1} . These complexes naturally admit a dual graph which we use to compute maximum flows and minimum cuts (both topological and combinatorial). We show that a maximum flow in a simplicial complex can be found by solving a shortest paths problem in its dual graph. This idea was used by Hassin to solve the maximum flow problem in planar graphs [13]. Further, we show that finding a minimum topological cut can be done by finding a minimum cost circulation in its dual graph. By setting the demand and capacity constraints equal to one in the minimum cost circulation problem we obtain an algorithm computing a minimum combinatorial cut.

Maximum flows in graphs can be computed using the Ford-Fulkerson algorithm. Moreover, the fact that the Ford-Fulkerson algorithm halts serves as a proof that there exists a maximum integral flow when the graph has integral capacity constraints. In dimensions $d > 1$ the maximum flow may be fractional, even with integral capacity constraints. The problem arises due to the existence of torsion in simplicial complexes of dimension $d > 1$. We show that despite the maximum flow being fractional the Ford-Fulkerson algorithm halts on simplicial complexes. However, in order for it to halt a special heuristic on picking the high dimensional analog of an augmenting path must be implemented. Despite the algorithm halting it could we could not prove a polynomial upper bound on the number of iterations it takes.

2 Preliminaries

Given a simplicial complex \mathcal{K} we define $C_d(\mathcal{K}, G)$, $Z_d(\mathcal{K}, G)$, $B_d(\mathcal{K}, G)$, and $H_d(\mathcal{K}, G)$ to be the d -dimensional chain, cycle, boundary, and homology spaces with coefficients over G . In this paper we will consider coefficients over \mathbb{R} and \mathbb{Z} , however when working over \mathbb{R} we will typically drop the G in the notation; for example $C_d(\mathcal{K})$ will refer to the d th chain group over the reals. We will use ∂_d and δ_d to denote the d -dimensional boundary and coboundary operators and we will use \mathcal{K}^d to denote the d -skeleton of \mathcal{K} . We view chains and cochains as both vectors and as functions which are equivalent viewpoints up to duality. For any chain σ by $\text{supp}(\sigma)$ we denote its support which is the set of simplices given a non-zero value on the chain. We call a $(d-1)$ -chain *null-homologous* if it is the boundary of some d -chain.

The fundamental theorem of finitely generated abelian groups gives us the decomposition $H_d(\mathcal{K}, \mathbb{Z}) \cong \mathbb{Z}^k \oplus \mathbb{Z}_{t_1} \oplus \cdots \oplus \mathbb{Z}_{t_n}$ for some $k \in \mathbb{N}$. We call the subgroup $\mathbb{Z}_{t_1} \oplus \cdots \oplus \mathbb{Z}_{t_n}$ the *torsion subgroup* of $H_d(\mathcal{K}, \mathbb{Z})$ and when this subgroup is trivial we call the complex *torsion-free*. We say \mathcal{K} is relatively torsion-free in dimension d if the relative homology groups $H_d(\mathcal{L}, \mathcal{L}_0, \mathbb{Z})$ is torsion-free for all subcomplexes \mathcal{L} and \mathcal{L}_0 of dimensions d and $d-1$, respectively. There exist complexes that are torsion-free but are not relatively torsion-free; see [5] for examples.

Let A be a matrix; we say that A is *totally unimodular* if every square submatrix A' of A has $\det(A') \in \{-1, 0, 1\}$. Totally unimodular matrices are important in combinatorial optimization because linear programs with totally unimodular constraint matrices are guaranteed to have integral solutions [11]. Dey, Hirani, and Krishnamoorthy have provided topological conditions on when a simplicial complex has a totally unimodular boundary matrix [5] stated below. The d th boundary matrix ∂_d of a simplicial complex is totally unimodular if and only if the complex is relative torsion-free in dimension $d-1$.

► **Theorem 1** (Dey et al. [5], Theorem 5.2). *Let \mathcal{K} be a d -dimensional simplicial complex. The boundary matrix $\partial_d: C_d(\mathcal{K}) \rightarrow C_{d-1}(\mathcal{K})$ is totally unimodular if and only if $H_{d-1}(\mathcal{L}, \mathcal{L}_0, \mathbb{Z})$ is torsion-free for all pure subcomplexes $\mathcal{L}_0, \mathcal{L}$ of \mathcal{K} of dimensions $d-1$ and d where $\mathcal{L}_0 \subset \mathcal{L}$.*

Throughout this paper we utilize discrete Hodge theory and recommend the survey by Lim [20] as an introduction to the topic. In particular, we use the Hodge decomposition which can be stated as a result on real valued matrices satisfying $AB = 0$.

► **Theorem 2** (Hodge decomposition [20]). *Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ be matrices satisfying $AB = 0$. We can decompose \mathbb{R}^n into the orthogonal direct sum $\mathbb{R}^n = \text{im}(A^T) \oplus \ker(A^T A + B B^T) \oplus \text{im}(B)$.*

Setting $A = \partial_d$ and $B = \partial_{d+1}$ yields the Hodge decomposition for simplicial complexes. The middle term of the direct sum becomes $\ker(\delta_{d+1}\partial_{d+1} + \partial_d\delta_d)$. The linear operator $\delta_{d+1}\partial_{d+1} + \partial_d\delta_d$ is known as the combinatorial Laplacian of \mathcal{K} which is a generalization of the graph Laplacian. Moreover, it can be shown that $\ker(\delta_{d+1}\partial_{d+1} + \partial_d\delta_d) \cong H_d(\mathcal{K}, \mathbb{R})$. We now state the Hodge decomposition on simplicial complexes as the following isomorphism $C_d(\mathcal{K}, \mathbb{R}) \cong \text{im}(\partial_d) \oplus H_d(\mathcal{K}, \mathbb{R}) \oplus \text{im}(\partial_{d+1})$.

3 Flows and cuts

In this section we give an overview of our generalizations of flows and cuts from graphs to simplicial complexes. Flows and cuts in higher dimensional settings have been studied previously. Duval, Klivans, and Martin have generalized cuts and flows to the setting of

CW complexes [7]. Their definitions are algebraic; defining cuts to be elements of $\text{im}(\delta)$ and flows to be elements of $\ker(\partial)$. Our definitions are closely related, but are motivated by the algorithmic problems of computing max-flows and min-cuts. In Section 3.1 we give definitions of flows and cuts from the perspective of algebraic topology, and in Section 3.2 we give a combinatorial definition of a cut in a simplicial complex. The distinction between the two types of cuts will be important when formulating the minimum cut problem on simplicial complexes.

3.1 Topological flows and cuts

First we briefly recall the definition of an st -flow in a directed graph $G = (V, E)$. An st -flow f is a function $f: E \rightarrow \mathbb{R}$ satisfying the conservation of flow constraint: for all $v \in V \setminus \{s, t\}$ we have $\sum_{(u,v) \in E} f(u, v) = \sum_{(v,u) \in E} f(v, u)$. That is, the amount of flow entering the vertex equals the amount of flow leaving the vertex. The value of f is equal to the amount of flow leaving s (or equivalently, entering t). Alternatively, we may view f as a 1-chain and we have $\partial f = k(t - s)$ where k is the value of f . Note that $t - s$ is a null-homologous 0-cycle. More generally, for any null-homologous $(d - 1)$ -cycle γ we call a d -chain f with $\partial f = k\gamma$ a γ -flow of value k . Note that under our naming convention an “ st -flow” in a graph would be called a $(t - s)$ -flow. However, in the case of graphs we use the traditional naming convention and call a flow from s to t an st -flow.

► **Definition 3** (γ -flow). *Let \mathcal{K} be a d -dimensional simplicial complex and γ be a null-homologous $(d - 1)$ -cycle in \mathcal{K} . A γ -flow is a d -chain f with $\partial f = k\gamma$ where $k \in \mathbb{R}$. We call k the value of the flow f and denote the value of f by $\|f\|$. We say that f is feasible with respect to a capacity function $c: \mathcal{K}^d \rightarrow \mathbb{R}^+$ if $0 \leq f(\sigma) \leq c(\sigma)$ for all $\sigma \in \mathcal{K}^d$.*

Our definition of a γ -flow is very close to the algebraic definition which is element of $\ker(\partial)$. Given a simplicial complex \mathcal{K} and a γ -flow f of value k we convert f into a *circulation*, where a circulation is defined to be an element of $\ker(\partial)$. To convert f into a circulation we add an additional basis element to $C_d(\mathcal{K})$, call it Σ , whose boundary is $\partial \Sigma = -\gamma$. This operation is purely algebraic; we should think of it as operating on the chain complex rather than the underlying topological space. Now we construct the circulation $f' = f + k\Sigma$. We call any circulation built from a γ -flow a γ -circulation. Clearly, $f' \in \ker(\partial)$ in the new chain complex. Moreover, there is a clear bijection between γ -flows and γ -circulations. The value of the circulation is the value of $f'(\Sigma)$, so this bijection preserves the value.

We now shift our focus to the generalization of cuts to a simplicial complex. The algebraic definition, elements of $\text{im}(\delta)$, is natural. The cut space of a graph is commonly defined to be the space spanned by the coboundaries of each vertex. In a simplicial complex \mathcal{K} , removing the support of a d -chain in $\text{im}(\delta)$ increases $\dim H_{d-1}(\mathcal{K})$. In a graph G , removing the support of any 1-chain in $\text{im}(\delta)$ increases $\dim H_0(G)$ which is equivalent to increasing the number of connected components of G .

The above definition implies that a cut is a d -chain in a d -dimensional simplicial complex. However, for our purposes we will define a cut to be a $(d - 1)$ -cochain. To motivate our definition we recall the notion of an st -cut in a graph. An st -cut in a graph is a partition of the vertices into sets S and T such that $s \in S$ and $t \in T$. Define $p: V(G) \rightarrow \{0, 1\}$ such that $p(v) = 1$ if $v \in S$ and $p(v) = 0$ if $v \in T$. The support of the coboundary of p is a set of edges whose removal destroys all st -paths. That is, upon removing the support, the 0-cycle $t - s$ is no longer null-homologous. Moreover, p is a 0-cochain with $p(t - s) = -1$. The sign of $p(t - s)$ will be important when we consider directed cuts. With this in mind we define our notion of a γ -cut.

► **Definition 4** (γ -cut). Let \mathcal{K} be a d -dimensional simplicial complex with weight function $c: \mathcal{K}^d \rightarrow \mathbb{R}^+$ and γ be a null-homologous $(d-1)$ -cycle in \mathcal{K} . A γ -cut is a $(d-1)$ -cochain p such that $p(\gamma) = -1$. Denote the coboundary of p as the formal sum $\delta(p) = \sum \alpha_i \sigma_i$, we define the size of a γ -cut to be $\|p\| = \sum |\alpha_i c(\sigma_i)|$.

Because of the requirement that $p(\gamma) = -1$ we call p a *unit* γ -cut. By relaxing this requirement to $p(\gamma) < 0$ the cochain p still behaves as a γ -cut, but its size can become arbitrarily small by multiplying by some small value $\epsilon > 0$. We justify our definition with the following proposition which shows that removing the support of the coboundary of a γ -cut prevents γ from being null-homologous.

► **Proposition 5.** Let \mathcal{K} be d -dimensional simplicial complex and p be a γ -cut. The cycle γ is not null-homologous in the subcomplex $\mathcal{K} \setminus \text{supp}(\delta(p))$.

3.2 Combinatorial cuts

Alternatively, we can view a γ -cut as a discrete set of d -simplices rather than a d -chain. In the case of graphs a combinatorial st -cut is just a set of edges whose removal disconnects s from t . This distinction will become important when we consider the minimization problem of finding a minimum cost set of d -simplices whose removal prevents γ from being null-homologous.

► **Definition 6** (Combinatorial γ -cut). Let \mathcal{K} be a d -dimensional simplicial complex with weight function $c: \mathcal{K}^d \rightarrow \mathbb{R}^+$ and γ be a null-homologous $(d-1)$ -cycle in \mathcal{K} . A **combinatorial γ -cut** is a set of d -simplices $C \subseteq \mathcal{K}^d$ such that γ is not null-homologous in $\mathcal{K} \setminus \text{supp}(C)$. The size of a combinatorial γ -cut is defined by the sum of the weights of the d -simplices $\|C\| = \sum_{\sigma \in C} c(\sigma)$.

The next proposition shows a relationship between γ -cuts and combinatorial γ -cuts. Removing a combinatorial γ -cut C from \mathcal{K} increases $\dim H_{d-1}(\mathcal{K})$. This is because removing C must decrease the rank of ∂_d and by duality this also decreases the rank of δ_d which increases the dimension of $H^{d-1}(\mathcal{K}) \cong H_{d-1}(\mathcal{K})$. It follows that C must contain the support of some coboundary. Given an additional minimality condition on C we show that C is equal to the support of some coboundary.

► **Proposition 7.** Let C be a combinatorial γ -cut in a d -dimensional simplicial complex \mathcal{K} . Further, assume that C is minimal in the sense that for all $C' \subset C$ we have $\dim H_{d-1}(\mathcal{K} \setminus C') < \dim H_{d-1}(\mathcal{K} \setminus C)$. There exists a $(d-1)$ -cochain p such that $\text{supp}(\delta(p)) = C$.

In graphs the linear program solving the minimum st -cut problem takes as input a directed graph and returns a set of directed edges whose removal destroys all directed st -paths. This is called a directed cut. After removing the directed cut the 0-cycle $t - s$ may still be null-homologous; we can find a 1-chain with boundary $t - s$ using negative coefficients to traverse an edge in the backwards direction. In order to generalize the minimum cut linear program to simplicial complexes we will need to define a directed combinatorial γ -cut, which requires the additional assumption that the d -simplices of \mathcal{K} are oriented.

► **Definition 8** (Directed combinatorial γ -cut). Let \mathcal{K} be an oriented d -dimensional simplicial complex with weight function $c: \mathcal{K}^d \rightarrow \mathbb{R}$ and γ be a null-homologous $(d-1)$ -cycle in \mathcal{K} . A **directed combinatorial γ -cut** is a set of d -simplices $C \subset \mathcal{K}^d$ such that in $\mathcal{K} \setminus \text{supp}(C)$ there exists no d -chain Γ with non-negative coefficients such that $\partial \Gamma = \gamma$. The size of a directed combinatorial γ -cut is defined by the sum of the weights of the d -simplices $\|C\| = \sum_{\sigma \in C} c(\sigma)$.

Given a directed graph consider an st -cut given by the cochain definition. That is, a 0-cochain $p: V(G) \rightarrow \{0, 1\}$ with $p(s) = 1$ and $p(t) = 0$ partitioning V into S and T . The support of $\delta(p)$ consists of two types of edges: edges leaving S and entering T , and edges leaving T and entering S . If $e \in E$ leaves S and enters T we have $(p \circ \partial)(e) = -1$ and if e leaves T and enters S we have $(p \circ \partial)(e) = 1$. To construct a directed st -cut we simply take all of the edges mapped to -1 . The following proposition shows that we can build a directed combinatorial γ -cut from a coboundary just like in the case of directed graphs.

► **Proposition 9.** *Let p be a γ -cut with coboundary $\delta(p) = \sum \alpha_i \sigma_i$. The set of d -simplices $C = \{\sigma_i \mid \alpha_i < 0\}$ is a directed combinatorial γ -cut.*

To conclude the section we will show that computing a minimum combinatorial γ -cut is NP-hard. As we will see in Section 4 minimum topological γ -cuts can be computed with linear programming. Our hardness result holds for both the directed and undirected cases. Our hardness result is a reduction from the well-known NP-hard hitting set problem which we will now define. Given a set S and a collection of subsets $\Sigma = (S_1, \dots, S_n)$ where $S_i \subseteq S$ the hitting set problem asks to find the smallest subset $S' \subseteq S$ such that $S' \cap S_i \neq \emptyset$ for all S_i . We call such a subset S' a *hitting set* for (S, Σ) .

► **Theorem 10.** *Let \mathcal{K} be a d -dimensional simplicial complex and γ be a null-homologous $(d - 1)$ -cycle. Computing a minimum combinatorial γ -cut is NP-hard for $d \geq 2$.*

Proof. Our proof is a reduction from the hitting set problem. First we consider the case when $d = 2$ then we generalize to any $d \geq 2$. Let S be a set and $\Sigma = (S_1, \dots, S_n)$ where each $S_i \subseteq S$. We construct a 2-dimensional simplicial complex \mathcal{K} from S and Σ in the following way. For each $S_i \in \Sigma$ construct a triangulated disk \mathcal{D}_i such that $\partial \mathcal{D}_i = \gamma$. That is, each \mathcal{D}_i shares the common boundary γ . To accomplish this we construct each \mathcal{D}_i by beginning with a single triangle t with $\partial t = \gamma$ and repeatedly adding a new vertex in the center of some triangle with edges connecting it to every vertex in that triangle. By this process we can construct a disk containing any odd number of triangles as each step increments the number of triangles in the disk by two. Moreover, at each step the boundary of the disk is always γ . We construct each disk \mathcal{D}_i such that \mathcal{D}_i consists of one triangle $t_{i,s}$ for each element $s \in S_i$ and potentially one extra triangle t'_i in the case that $|S_i|$ is even. Next, for each $s \in S$ and S_i with $s \in S_i$, we construct the quotient space by identifying each $t_{i,s}$ into a single triangle. A minimum combinatorial γ -cut C must contain at least one triangle from each \mathcal{D}_i and without loss of generality we can assume C does not contain any t'_i . If $t'_i \in C$ then by minimality it is the only triangle in $C \cap \text{supp}(\mathcal{D}_i)$ and we can swap it with any other triangle in \mathcal{D}_i without increasing the size of the cut. By construction C is a hitting set for (S, Σ) since each $C \cap \text{supp}(\mathcal{D}_i) \neq \emptyset$ for all \mathcal{D}_i . The proof generalizes to $d > 2$ by generalizing the subdivision processes. ◀

4 Linear programming

4.1 Max-flow min-cut

A *simplicial flow network* is a tuple (\mathcal{K}, c, γ) where \mathcal{K} is an oriented d -dimensional simplicial complex, c is the *capacity function* which is a non-negative function $c: \mathcal{K}^d \rightarrow \mathbb{R}^+$, and γ is a null-homologous $(d - 1)$ -cycle. In a simplicial flow network we work with real coefficients; that is, we consider the chain groups $C_k(\mathcal{K}, \mathbb{R})$. In order to utilize the Hodge decomposition (Theorem 2) in a convenient way we modify $C_d(\mathcal{K})$ by adding an additional basis element Σ such that $\partial \Sigma = -\gamma$. Moreover, we extend the capacity function such that $c(\Sigma) = \infty$. This

allows us to work with circulations instead of flows while leaving the solution unchanged. The notation n_d will refer to the number of basis elements in $C_d(\mathcal{K}, \mathbb{R})$ which is now one more than the number of d -simplices in the underlying simplicial complex.

The goal of the maximum flow problem is to find a d -chain f obeying the capacity constraints such that $\partial f = k\gamma$ where $k \in \mathbb{R}$ is maximized. Equivalently, we find a d -cycle f which maximizes $f(\Sigma)$. The linear program for the max-flow problem in a simplicial flow network is identical to the familiar linear program for graphs, but expressed in terms of the coboundary operator. In a graph, conservation of flow at a vertex v is the constraint $\delta_1(v) \cdot f = 0$; to formulate the linear program in higher dimensions we simply replace vertices with $(d-1)$ -simplices. The Hodge decomposition states that cycles are orthogonal to coboundaries, so conservation of flow ensures that f is indeed a cycle. We now state the linear program for max-flow in a simplicial flow network.

$$\begin{aligned} & \text{maximize} && f(\Sigma) \\ & \text{subject to} && \delta(\tau) \cdot f = 0 \quad \text{for each } \tau \in \mathcal{K}^{d-1} \\ & && 0 \leq f(\sigma) \leq c(\sigma) \quad \text{for each } \sigma \in \mathcal{K}^d \end{aligned} \tag{LP1}$$

We dualize LP1 to obtain a generalization of the minimum cut problem in directed graphs. To make the dualization more explicit we will write out LP1 in matrix form: maximize $s \cdot f$

subject to $Af \leq b$ and $f \geq 0$, where we have $A = \begin{bmatrix} \partial \\ -\partial \\ I_{n_d} \end{bmatrix}$, $b = \begin{bmatrix} 0_{n_{d-1}} \\ 0_{n_{d-1}} \\ c \end{bmatrix}$, $s = \begin{bmatrix} 0_{n_d-1} \\ 1 \end{bmatrix}$. The

matrix A has dimension $(2n_{d-1} + n_d) \times n_d$. In our notation I_k is the $k \times k$ identity matrix and 0_k is the $k \times 1$ column vector consisting of all zeros. Since the value of the flow is equal to $f(\Sigma)$ the vector s is all zeros except for the final entry which is indexed by Σ and receives an entry equal to one. Further, c is the $n_d \times 1$ capacity vector indexed by the d -simplices such that the entry indexed by σ has value equal to $c(\sigma)$.

We can now state the dual program in matrix form: minimize $y \cdot b$ subject to $y^T A \geq s$ and $y \geq 0$. The vector y is a $(2n_{d-1} + n_d) \times 1$ column vector indexed by both the $(d-1)$ -simplices and the d -simplices. However, only the entries indexed by d -simplices contribute to the objective function since b is zero everywhere outside of the capacity constraints. We will denote the truncated vector consisting of entries indexed by d -simplices by y_d and the entry corresponding to the d -simplex $\sigma \in \mathcal{K}^d$ will be denoted by $y_d(\sigma)$. Similarly we have two truncated vectors y_{d-1}^1 and y_{d-1}^2 corresponding to the entries indexed by the $(d-1)$ -simplices. Moreover, the rows of $y^T A \geq s$ are in the form $(y_{d-1}^1 - y_{d-1}^2)^T \partial + y_d \geq s$. For simplicity we define $y_{d-1} = y_{d-1}^1 - y_{d-1}^2$ and write $y_{d-1}(\tau)$ for the entry indexed by the $(d-1)$ -simplex τ . Putting this together, we state the dual linear program as follows.

$$\begin{aligned} & \text{minimize} && \sum_{\sigma \in \mathcal{K}^d} y_d(\sigma) c(\sigma) \\ & \text{subject to} && y_{d-1} \cdot \partial \sigma + y_d(\sigma) \geq 0 \quad \text{for each } \sigma \in \mathcal{K}^d \\ & && y_{d-1} \cdot \partial \Sigma + y_d(\Sigma) = 1, y_d \geq 0 \end{aligned} \tag{LP2}$$

Note the strict equality in the second constraint does not follow from the duality. However, we can assume a strict equality since if $y_{d-1} \cdot \partial \Sigma + y_d(\Sigma) > 1$ we can multiply $[y_{d-1}, y_d]^T$ by some scalar $\epsilon < 1$ to make the inequality tight. This multiplication only decreases the value of $\sum y_d(\sigma) c(\sigma)$ so it does not change the optimal solution.

In the case of graphs LP2 has dual variables for vertices and edges. Moreover, there exists an integral solution such that each vertex is either assigned a 0 or a 1 since a graph cut is a partition of the vertices. The second inequality requires $y_0(s) = 1$ and $y_0(t) = 0$. To see this, when solving an st -cut on a graph, the basis element Σ is an edge with $\partial\Sigma = s - t$, and $y_1(\Sigma) = 0$ otherwise the solution is infinite. This naturally defines a partition of the vertices: S containing vertices assigned a 1, and T containing vertices assigned a 0. The constraints force an edge to be assigned a 1 if it leaves S and enters T , otherwise it is assigned a 0. This solution can be interpreted as a 0-cochain p with $p(st) = 1$, or in the notation of our definition of a simplicial cut: $p(t - s) = -1$. Further, $y_1(e) = 1$ for every edge e that is negative on $\delta(p)$ and a 0 otherwise, hence y_1 fits our definition of a directed st -cut in a 1-complex. We will show the same result holds in higher dimensions; that is, y_d is a directed γ -cut arising from the $(d - 1)$ -cochain y_{d-1} .

► **Lemma 11.** *Let $y = [y_{d-1}, y_d]^T$ be an optimal solution to LP2. The set $\text{supp}(y_d)$ is a directed combinatorial γ -cut.*

► **Lemma 12.** *Let p be a γ -cut with coboundary $\delta(p) = \sum \alpha_i \sigma_i$ and let $\delta(p)^- = \sum_{\alpha_i < 0} \alpha_i \sigma_i$. The vector $[p, -\delta(p)^-]^T$ is a finite feasible solution to LP2.*

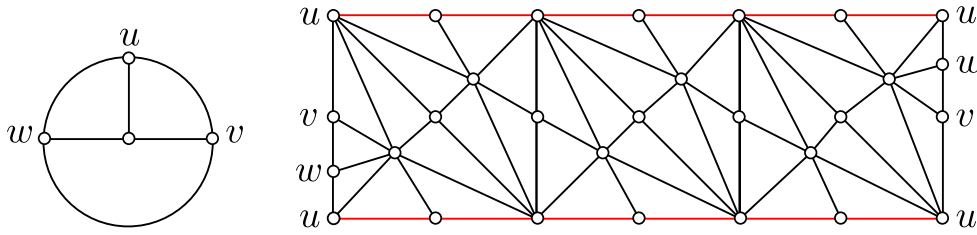
Lemma 11 tells us that a solution to LP2 yields a directed combinatorial γ -cut. Recall, by Proposition 9 every γ -cut p yields a directed combinatorial γ -cut by taking the coboundary $\delta(p) = \sum \alpha_i \sigma_i$ and considering the negative components $\delta(p)^- = \{\sigma_i \mid \alpha_i < 0\}$. By Lemma 12 $\delta(p)^-$ is a feasible solution to LP2; the cost of this solution is $c \cdot \delta(p)^-$. The coefficients α_i need not always equal one; hence in general we have $\|C\| \neq c \cdot \delta(p)^-$. It follows that LP2 need not return a minimum directed combinatorial γ -cut. In Theorem 15 we will give conditions describing when LP2 returns a directed combinatorial γ -cut. To conclude the section we state our main theorem about LP2 whose proof is immediate from Lemmas 11 and 12.

► **Theorem 13.** *Let $y = [y_{d-1}, y_d]^T$ be an optimal solution to LP2. The set $\text{supp}(y_d)$ is a directed combinatorial γ -cut such that $y_d = \delta(y_{d-1})^-$. Moreover, y_d minimizes $c \cdot \delta(p)^-$ where p ranges over all γ -cuts.*

4.2 Integral solutions

In this section we provide an example of a simplicial flow network with integral capacity constraints and fractional maximum flow. By Theorem 1 such a network must contain some relative torsion. This is achieved by the inclusion of a Möbius strip in our simplicial flow network. Our example will be used later in Theorem 14 showing that computing a maximum integral flow in a simplicial flow network is NP-hard.

We will now explicitly describe a simplicial flow network with integral capacities whose maximum flow value is fractional. Let \mathcal{M} be a triangulated Möbius strip with boundary $\partial\mathcal{M} = 2\alpha + \gamma$ such that two vertices in α have been identified making α a simple cycle. This identification makes γ a figure-eight. Now let \mathcal{D} be a triangulated disk oriented such that $\partial\mathcal{D} = -\alpha$. Call the resulting complex \mathcal{MD} . See Figure 1 for an illustration. The capacity function c has $c(t) = 1$ for each triangle $t \in \mathcal{MD}$. Now we solve the max-flow problem on $(\mathcal{MD}, c, \gamma)$. Note that for any flow f we have $f(t_1) = f(t_2)$ for all triangles $t_1, t_2 \in \mathcal{M}$; moreover, for all $t_1, t_2 \in \mathcal{D}$ we also have $f(t_1) = f(t_2)$. The value of any flow f on $(\mathcal{MD}, c, \gamma)$ is equal to its value on \mathcal{M} , and in order to maintain conservation of flow we must have $f(\mathcal{D}) = 2f(\mathcal{M})$. Now, the capacity constraints imply that the maximum flow f has $f(\mathcal{M}) = 1/2$ and $f(\mathcal{D}) = 1$. We have $\partial f = \frac{1}{2}\partial\mathcal{M} + \partial\mathcal{D} = \frac{1}{2}\gamma + \alpha - \alpha$. Hence, $\|f\| = 1/2$.



■ **Figure 1** A triangulated disk \mathcal{D} (left) and Möbius strip \mathcal{M} (right). The Möbius strip has two points on its boundary identified forming the vertex u . In red we have the input cycle (a figure-eight) γ and we set $\alpha = u w v u$. We orient the complex such that $\partial \mathcal{M} = \gamma + 2\alpha$ and $\partial \mathcal{D} = -\alpha$. The capacity on each simplex in both the disk and Möbius strip is one.

Maximum integral flow. Given a simplicial flow network (\mathcal{K}, c, γ) with integral capacities we consider the problem of finding the maximum integral flow. That is, a d -chain $f \in C_d(\mathcal{K}, \mathbb{Z})$ obeying the capacity constraints such that $\partial f = k\gamma$ where $k \in \mathbb{Z}$ is maximized. We show the problem is NP-hard by a reduction from graph 3-coloring. Our reduction is inspired by a MathOverflow post from Sergei Ivanov showing that finding a subcomplex homeomorphic to the 2-sphere is NP-hard [15]. Given a graph G we construct a 2-dimensional simplicial flow network whose maximum flow is integral if and only if G is 3-colorable.

► **Theorem 14.** *Let (\mathcal{K}, c, γ) be a simplicial flow network where \mathcal{K} is a 2-complex and c is integral. Computing a maximum integral flow of (\mathcal{K}, c, γ) is NP-hard.*

Proof. Let $G = (V, E)$ be a graph. We will construct a simplicial flow network (\mathcal{K}, c, γ) such that its maximum flow is integral if and only if G is 3-colorable.

We start our construction with a punctured sphere \mathcal{S} containing $|V| + 1$ boundary components called γ and β_v for each $v \in V$. For each boundary component β_v we construct three disks R_v, B_v, G_v each with boundary $-\beta_v$. These disks represent the three colors in our coloring: red, blue, and green. We refer to these disks as *color disks* and use \mathcal{C}_v to denote an arbitrary color disk associated with v and use $k \in \{r, b, g\}$ to denote an arbitrary color. On each color disk \mathcal{C}_v we add a boundary component for each edge $e = (u, v)$ incident to v . By $\beta_{v,e,k}$ we denote the boundary component corresponding to the vertex v , edge e , and color k . For each edge $e = (u, v)$ and each pair of boundary components β_{u,e,k_u} and β_{v,e,k_v} with $k_u \neq k_v$ we construct a tube with boundary components $-\beta_{u,e,k_u}$ and $-\beta_{v,e,k_v}$ denoted \mathcal{T}_{e,k_u,k_v} . When $k_u = k_v = k$ we construct a tube $\mathcal{T}_{e,k,k}$ and puncture it with a third boundary component α and construct a negatively oriented real projective plane $\mathcal{RP}_{e,k}$ with boundary $\partial \mathcal{RP}_{e,k} = -2\alpha$. We call the resulting complex \mathcal{K} and assign a capacity $c(\sigma) = 1$ for every triangle σ in \mathcal{K} .

We will show that a maximum integral flow f of \mathcal{K} has $\|f\| = 1$ if and only if G is 3-colorable. The following four properties of a maximum integral flow f imply that G is 3-colorable.

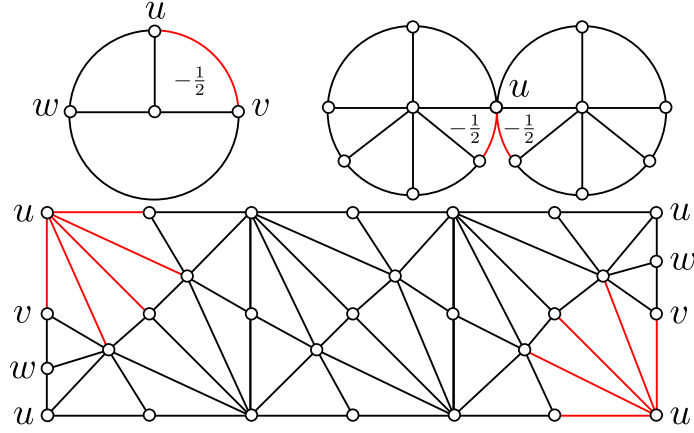
- f must assign exactly one unit of flow to each triangle in \mathcal{S} since the value of f is equal to $f(\mathcal{S})$.
- For each vertex $v \in V$ exactly one color disk \mathcal{C}_v is assigned one unit of flow while the other two color disks associated with v are assigned zero units of flow. Otherwise, either conservation of flow is violated or some color disk is assigned a fractional flow value.
- For each edge $e = (u, v) \in E$ exactly one tube \mathcal{T}_{e,k_u,k_v} with $k_u \neq k_v$ must be assigned one unit of flow with all other tubes associated with e assigned zero units of flow. The tube \mathcal{T}_{e,k_u,k_v} assigned one unit of flow is the tube connecting the color disks \mathcal{C}_v and \mathcal{C}_u that are assigned one unit of flow by the previous property.

- f assigns zero flow to every $\mathcal{T}_{e,k,k}$ and $\mathcal{RP}_{e,k}$ since otherwise the triangles in $\mathcal{RP}_{e,k}$ would need to have $1/2$ units of flow assigned to them to maintain conservation of flow.

These four properties imply that the set of color disks $\{\mathcal{C}_v \mid f(\sigma) = 1, \forall \sigma \in \mathcal{C}_v\}$ corresponds to a 3-coloring of G . Conversely, given a 3-coloring of G we assign a flow value of one to each color disk corresponding to the 3-coloring. We extend this assignment to a γ -flow of value one by assigning a flow value of one to \mathcal{S} and the tubes corresponding to the 3-coloring. ◀

Integral cuts. The goal of this section is to show that for simplicial complexes that are relative torsion-free in dimension $d - 1$ there exists optimal solutions to LP2 whose support is a minimum combinatorial γ -cut. Note that by Theorem 1 a simplicial complex that is relative torsion-free in dimension $d - 1$ has a totally unimodular d -dimensional boundary matrix. The total unimodularity is key to our proof. However, we first provide an example of a complex (with relative torsion) whose optimal solution's support does not form a minimum combinatorial γ -cut. Our construction is a slight modification of \mathcal{MD} defined in Section 4.2.

Consider the simplicial complex constructed by taking \mathcal{MD} and glueing a wedge sum of two disks \mathcal{W} along the figure-eight γ . That is, $\partial\mathcal{W} = \gamma$. We give every triangle in the resulting complex a capacity equal to one. A maximum γ -flow has value $3/2$, so the dual program finds a γ -cut of the same value. One potential optimal solution is a $(d - 1)$ -cochain whose coboundary assigns a value of $-1/2$ to two triangles in \mathcal{W} and a value of $-1/2$ to one triangle in \mathcal{D} . The support of this coboundary has weight equal to three, however a minimal combinatorial γ -cut has weight two by taking only one triangle from \mathcal{W} and one from \mathcal{D} . See Figure 2 for an illustration.



■ **Figure 2** The simplicial complex \mathcal{MD} with a wedge sum of two disks \mathcal{W} identified to the figure-eight γ . In red we have a 1-cochain which assigns a value of $-1/2$ to each red edge. The coboundary of the red cochain assigns a value of $-1/2$ to one triangle in \mathcal{D} and a value of $-1/2$ to two triangles in \mathcal{W} . The value of the red cochain coincides with the value of the maximum γ -flow. However, its support is not a minimum combinatorial γ -cut. A minimum combinatorial γ -cut picks one triangle from \mathcal{D} and one triangle from \mathcal{W} .

Now, we show that when \mathcal{K} is relative torsion-free in dimension $d - 1$ LP2 has an optimal solution whose support is a minimum directed combinatorial γ -cut. Specifically, we show that a solution existing on a vertex of the polytope defined by the constraints of LP2 is a cochain y_{d-1} with negative coboundary y_d such that $y_d(\sigma) \in \{0, 1\}$ for all $\sigma \in \mathcal{K}^d$ hence $\sum y_d(\sigma)c(\sigma) = \|\text{supp}(y_d)\|$. That is, the value of a vertex solution to LP2 is equal to the cost of $\text{supp}(y_d)$ as a directed combinatorial γ -cut.

► **Theorem 15.** *Let \mathcal{K} be d -dimensional simplicial complex that is relative torsion-free in dimension $d - 1$ and $[y_{d-1}, y_d]^T$ be an optimal vertex solution to the dual program. The set $\text{supp}(y_d)$ is a minimum directed combinatorial γ -cut.*

Proof. We can write the constraint matrix of LP2 as the $2n_d \times (n_d + n_{d-1})$ block matrix

$$A = \begin{bmatrix} \delta & I_{n_d} \\ 0_{n_d} & I_{n_d} \end{bmatrix}.$$

Since \mathcal{K} is relative torsion-free in dimension $d - 1$ Theorem 1 tells us that ∂_d is totally unimodular; further, we have that $\partial^T = \delta$ is also totally unimodular. Total unimodularity is preserved under the operation of adding a row or column consisting of exactly one component equal to 1 and the remaining components equal to 0, so A is totally unimodular [23, Section 19.4]. We write LP2 as the linear system $Ax \geq b$ where b is a $n_d + n_{d-1}$ dimensional vector with exactly one component equal to 1 and the remaining components equal to 0. Let $y = [y_{d-1}, y_d]^T$ be an optimal vertex solution to LP2. For every $(d - 1)$ -simplex $\tau \in \mathcal{K}^{d-1}$ we either have $y_{d-1}(\tau) \geq 0$ or $y_{d-1}(\tau) \leq 0$. Let $I'_{n_{d-1}}$ be the matrix whose rows correspond to these inequalities. Note that $I'_{n_{d-1}}$ is a diagonal matrix with entries in $\{-1, 1\}$. Now we consider the $(2n_d + n_{d-1}) \times (n_d + n_{d-1})$ dimensional linear system $A'x \geq b'$ where

$$A' = \begin{bmatrix} \delta & I_{n_d} \\ 0 & I_{n_d} \\ I'_{n_{d-1}} & 0 \end{bmatrix}$$

and b' is constructed by appending extra zeros to b . We construct y' from y similarly. Note that A' is totally unimodular and y' is a vertex solution of the system. There exists a vertex v of the polyhedron $P \subseteq \mathbb{R}^{n_d + n_{d-1}}$ corresponding to the linear system such that $A'y' = v \geq b'$ such that $n_{d-1} + n_d$ constraints are linearly independent and tight. Hence, there is an $(n_{d-1} + n_d) \times (n_{d-1} + n_d)$ square submatrix A'' with $A''y' = b''$ where b'' is b' restricted to the tight constraints. We will use Cramer's rule to show that the vertex solution y has components coming from the set $\{-1, 0, 1\}$. Let $A''_{i,b''}$ be the matrix obtained by replacing the i^{th} column of A'' with b'' . By Cramer's rule we compute the i^{th} component of y as $y_i = \frac{\det(A''_{i,b''})}{\det(A'')}$. Since both $A''_{i,b''}$ and A'' are totally unimodular we have $v_i \in \{-1, 0, 1\}$. Further, we know that A'' is non-singular because it corresponds to linearly independent constraints.

By the above argument we know that an optimal solution y to LP2 has all of its components contained in the set $\{-1, 0, 1\}$. The constraint $y_d \geq 0$ means that for all d -simplices σ we have $y_d(\sigma) \in \{0, 1\}$ and $\sum_{\sigma \in \mathcal{K}^d} y_d(\sigma)c(\sigma) = \|\text{supp}(y_d)\|$. Hence, $\text{supp}(y_d)$ is a minimum directed combinatorial γ -cut. ◀

5 Embedded simplicial complexes

In this section we consider a simplicial flow network (\mathcal{K}, c, γ) where \mathcal{K} is a d -dimensional simplicial complex with an embedding into \mathbb{R}^{d+1} . Alexander duality implies that $\mathbb{R}^{d+1} \setminus \mathcal{K}$ consists of $\beta_d + 1$ connected components where $\beta_d = \dim H_d(\mathcal{K})$ is the d th Betti number. We call these connected components *voids*; exactly one void is unbounded and we denote the voids by V_i for $1 \leq i \leq \beta_{d+1}$. Given an embedding into \mathbb{R}^{d+1} , computing the voids of \mathcal{K} can be done in polynomial time [6]. Further, we assume that the d -simplices are consistently oriented with respect to the voids. The embedding guarantees that every d -simplex σ appears on the boundary of at most two voids; by our assumption if σ appears on the boundary of

two voids then it must be oriented positively on one and negatively on the other. We denote the boundary of the void V_i by $\text{Bd}(V_i)$. Every d -simplex contained in the support of some d -cycle is on the boundaries of exactly two voids; it follows that the boundaries of any set of β_d voids is a basis of $H_d(\mathcal{K})$.

In order to state our theorems we need one additional assumption on \mathcal{K} . We assume there exists some void V_i containing two unit γ -flows Γ_1, Γ_2 whose supports partition $\text{Bd}(V_i)$: $\text{supp}(\Gamma_1) \cap \text{supp}(\Gamma_2) = \emptyset$ and $\text{supp}(\Gamma_1) \cup \text{supp}(\Gamma_2) = \text{Bd}(V_i)$. This assumption makes our problem analogous to an st -flow network in a planar graph such that s and t appear on the same face. The existence of two unit γ -flows partitioning the boundary is analogous to the two disjoint st -paths on the boundary of the face. It will be convenient to take the negation of Γ_1 and treat it as a unit $(-\gamma)$ -flow; otherwise the assumption conflicts with the assumed consistent orientation. This is equivalent as it does not change the support of the flow, so for the rest of the section we will take Γ_1 to be a unit $(-\gamma)$ -flow.

From \mathcal{K} we construct its directed dual graph \mathcal{K}^* as follows. Each void becomes a vertex of \mathcal{K}^* . Each d -simplex on the boundary of two voids becomes an edge; since we assumed the d -simplices are consistently oriented we direct the dual edge from the negatively oriented void to the positively oriented void. The remaining d -simplices only appear on one void and become loops in \mathcal{K}^* . For a d -simplex σ on the boundary of voids u and v we denote its corresponding dual edge $\sigma^* = (u^*, v^*)$ and we weight the edges by the capacity function: $c^*(\sigma^*) = c(\sigma)$. Let v_i^* be the vertex dual to the void whose boundary is partitioned by $\text{supp}(\Gamma_1)$ and $\text{supp}(\Gamma_2)$. We split v_i^* into two new vertices denoted s^* and t^* . The edges incident to v_i^* whose dual d -simplices were contained in $\text{supp}(\Gamma_1)$ become incident to s^* , and the edges whose dual d -simplices were contained in $\text{supp}(\Gamma_2)$ become incident to t^* . We add the directed edge (t^*, s^*) and set its capacity to infinity; $c^*((t^*, s^*)) = \infty$. Returning to the analogy of a planar graph with s and t on the same face, splitting v_i^* is analogous to adding an additional edge from t to s which splits their common face into two. However, for our purposes we are only concerned with the algebraic properties of the construction and do not actually need to modify the simplicial complex.

We need to update the chain complex associated with \mathcal{K} to account for the voids and the splitting of v_i^* . We add an additional basis element Σ to $C_d(\mathcal{K})$ such that $\partial \Sigma = \gamma$ and give it infinite capacity; $c(\Sigma) = \infty$. In our construction Σ is dual to the edge (t^*, s^*) . In our planar graph analogy Σ plays the role of an edge from t to s drawn entirely in the outer face; to make this precise we will need to add an additional chain group $C_{d+1}(\mathcal{K})$. We add each void V_j with $j \neq i$ as a basis element of $C_{d+1}(\mathcal{K})$ and define the boundary operator as $\partial_{d+1} V_j = \sum_{\sigma \in \text{Bd}(V_j)} (-1)^{k_\sigma} \sigma$ where $k_\sigma = 0$ if σ is oriented positively on V_j and $k_\sigma = 1$ if σ is oriented negatively on V_j . Next we add additional basis elements S and T whose boundaries are defined by $\partial_{d+1} S = \Gamma_1 + \Sigma$ and $\partial_{d+1} T = \Gamma_2 - \Sigma$. The inclusion of $C_{d+1}(\mathcal{K})$ results in a valid chain complex since by definition the image of ∂_{d+1} under each basis element is a d -cycle. Moreover, in the new complex we have $H_d(\mathcal{K}) \cong 0$ since the boundaries of the voids generate $H_d(\mathcal{K})$.

Given our new chain complex we can extend the dual graph \mathcal{K}^* to a dual complex; this construction is reminiscent of the dual of a polyhedron. We define the dual complex by the isomorphism of chain groups $C_k(\mathcal{K}^*) \cong C_{d-k+1}(\mathcal{K})$. The dual boundary operator $\partial_k^*: C_k(\mathcal{K}^*) \rightarrow C_{k-1}(\mathcal{K}^*)$ is the coboundary operator δ_{d-k+2} , and the dual coboundary operator $\delta_k^*: C_{k-1}(\mathcal{K}^*) \rightarrow C_k(\mathcal{K}^*)$ is the boundary operator ∂_{d-k+2} . The primal boundary operator commutes with the dual coboundary operator, and the primal coboundary operator commutes with the dual boundary operator. We summarize the construction with the following commutative diagram.

$$\begin{array}{ccccccc}
C_{d+1}(\mathcal{K}) & \xleftarrow[\delta_{d+1}]{\partial_{d+1}} & C_d(\mathcal{K}) & \xleftarrow[\delta_d]{\partial_d} & \dots & \xleftarrow[\delta_1]{\partial_1} & C_0(\mathcal{K}) \\
\uparrow \cong & & \uparrow \cong & & & & \uparrow \cong \\
C_0(\mathcal{K}^*) & \xleftarrow[\partial_1^*]{\delta_1^*} & C_1(\mathcal{K}^*) & \xleftarrow[\partial_2^*]{\delta_2^*} & \dots & \xleftarrow[\partial_{d+1}^*]{\delta_{d+1}^*} & C_{d+1}(\mathcal{K}^*)
\end{array}$$

We now have enough structure to state our duality theorems. We show that computing a max-flow for (\mathcal{K}, c, γ) is equivalent to computing a shortest path from s^* to t^* in \mathcal{K}^* and that computing a minimum cost γ -cut p is equivalent to computing a minimum cost unit s^*t^* -flow in \mathcal{K}^* .

Max-flow / shortest path duality. We compute a shortest path from s^* to t^* in \mathcal{K}^* using a well-known shortest paths linear program. Details on the linear program can be found in [9].

$$\begin{array}{ll}
\text{maximize} & \text{dist}(t^*) \\
\text{subject to} & \text{dist}(s^*) = 0 \\
& \text{dist}(v^*) - \text{dist}(u^*) \leq c^*((u^*, v^*)) \quad \forall (u^*, v^*) \in E
\end{array} \tag{LP3}$$

The solution to LP3 is a function $\text{dist}: V(\mathcal{K}^*) \rightarrow \mathbb{R}$ which maps a vertex to its distance from s^* under the weight function c . By duality, dist is a $(d+1)$ -cochain mapping the voids to \mathbb{R} . In the following theorem we will show that dist is equivalent to a γ -flow with value equal to $\text{dist}(t^*)$.

► **Theorem 16.** *Let (\mathcal{K}, c, γ) be a simplicial flow network where \mathcal{K} is a d -dimensional simplicial complex embedded into \mathbb{R}^{d+1} with two unit γ -flows whose supports partition the boundary of some void $\text{Bd}(V_i)$. There is a bijection between γ -flows of (\mathcal{K}, c, γ) and s^*t^* -paths in \mathcal{K}^* such that the value of a γ -flow equals the length of its corresponding s^*t^* -path.*

Min-cut / min cost flow duality. We begin this section by stating the minimum cost flow problem in graphs. The minimum cost flow problem asks to find the cheapest way to send k units of flow from s to t . An instance of the minimum cost flow problem is a tuple (G, w, c, k) where $G = (V, E)$ is a directed graph, $w, c \in C_1(G)$, and $k \in \mathbb{R}$. The 1-chains represent the weight and capacity of each edge, and k is the demand of the network. The goal of the minimum cost flow problem is to find an st -flow satisfying both capacity and demand constraints. The demand constraint can be stated as $\delta(t) \cdot f = k$ and ensures that f sends exactly k units of flow from s to t . We will compute a minimum directed γ -cut in \mathcal{K} by solving the minimum cost flow problem with $k = 1$ in \mathcal{K}^* . We assume there is a weight function $w: \mathcal{K}^d \rightarrow \mathbb{R}^+$ on the d -skeleton of \mathcal{K} , which after dualizing becomes a weight function w^* on the edges of \mathcal{K}^* . In the following theorem the capacity function is not needed, so we will assume each edge in \mathcal{K}^* has infinite capacity.

► **Theorem 17.** *Let \mathcal{K} be a d -dimensional simplicial complex embedded into \mathbb{R}^{d+1} with two unit γ -flows whose supports partition the boundary of some void $\text{Bd}(V_i)$. There is a bijection between γ -cuts p in \mathcal{K} and unit s^*t^* -flows f in \mathcal{K}^* such that $\|p\| = \sum w^*(e)f(e)$.*

► **Corollary 18.** *Let \mathcal{K} be a d -dimensional simplicial complex embedded in \mathbb{R}^{d+1} with two unit γ -flows partitioning some $\text{Bd}(V_i)$. There is a polynomial time algorithm computing a minimum directed combinatorial γ -cut.*

Proof. We solve the minimum cost circulation problem in \mathcal{K}^* setting the demand and every capacity constraint equal to one. The resulting flow is dual to a γ -cut p in \mathcal{K} . Since the minimum cost circulation is integral we have $\|\text{supp}(\delta(p))\| = \|p\|$. That is, the cost of p as a γ -cut equals the cost of $\text{supp}(\delta(p))$ as a combinatorial γ -cut. ◀

6 Ford-Fulkerson algorithm

In this section we show how the Ford-Fulkerson algorithm can be used to compute a max-flow of simplicial flow network (\mathcal{K}, c, γ) . In a simplicial flow network the Ford-Fulkerson algorithm picks out a *augmenting chain* at every iteration which is a high dimensional generalization of an augmenting path. As shown in Section 4.2 a max-flow of a simplicial flow network with integral capacities may not be integral, so it is not immediate that Ford-Fulkerson is guaranteed to halt. To remedy this, our implementation of Ford-Fulkerson contains a heuristic reminiscent of the network simplex algorithm. Our heuristic guarantees that at every iteration of Ford-Fulkerson the flow is a solution on a vertex of the polytope defined by the linear program. Hence, our heuristic makes our implementation of Ford-Fulkerson into a special case of the simplex algorithm. It follows that Ford-Fulkerson does halt on a simplicial flow network, but the running time may be exponential. Our heuristic for picking augmenting chains takes $O(n^{\omega+1})$ time since it requires solving $O(n)$ linear systems, each taking $O(n^\omega)$ time using standard methods [16].

► **Definition 19** (Residual complex). *Let (\mathcal{K}, c, γ) be a simplicial flow network and f be a feasible flow on the network. We define a new simplicial flow network called the **residual complex** to be the tuple $(\mathcal{K}_f, c_f, \gamma)$ constructed in the following way. The d -skeleton of \mathcal{K}_f is the union $\mathcal{K}^d \cup -\mathcal{K}^d$, that is, for each d -simplex σ in \mathcal{K} we add an additional d -simplex $-\sigma$ whose orientation is opposite of σ . $\mathcal{K}_f^{d'} = \mathcal{K}^{d'}$ for dimensions $d' < d$. The **residual capacity function** $c_f: \mathcal{K}_f^d \rightarrow \mathbb{R}$ is given by $c_f(\sigma) = \begin{cases} c(\sigma) - f(\sigma) & \sigma \in \mathcal{K}^d \\ f(\sigma) & -\sigma \in \mathcal{K}^d \end{cases}$.*

► **Definition 20** (Augmenting chain). *Let \mathcal{K}_f be a residual complex for the simplicial flow network (\mathcal{K}, c, γ) . An **augmenting chain** is a d -chain $\Gamma \in C_d(\mathcal{K}_f)$ such that $\Gamma = \sum \alpha_i \sigma_i$ and $\partial \Gamma = \gamma$ with $\alpha_i \geq 0$.*

Note that an augmenting chain need not obey the residual capacity constraint c_f . This is because after finding an augmenting chain the amount of flow sent through the chain will be normalized by the coefficients α_i producing a new chain respecting the capacity constraints. We now state the main theorem of the section.

► **Theorem 21.** *Let (\mathcal{K}, c, γ) be a simplicial flow network. A flow f is a maximum flow if and only if \mathcal{K}_f contains no augmenting chains.*

Augmenting chain heuristic. In this section we provide a heuristic for the Ford-Fulkerson algorithm that is guaranteed to halt on a simplicial flow network. Our example in Section 4.2 shows that a maximum flow may have fractional value, so it's not immediately clear that Ford-Fulkerson halts on all simplicial flow networks. To remedy this our heuristic ensures that at each step the flow corresponds to a vertex of the flow polytope (defined in the next paragraph). As there are a finite number of vertices, and the value of the flow increases at every step, it follows that under this heuristic Ford-Fulkerson must halt. Under our heuristic Ford-Fulkerson becomes a special case of the simplex algorithm. Our heuristic is reminiscent of the network simplex algorithm which maintains a tree at every iteration. See the book by Ahuja, Magnanti, and Orlin for an overview of the network simplex algorithm [1].

We define the *flow polytope* of (\mathcal{K}, c, γ) to be the polytope $P \subset \mathbb{R}^{n_d}$ defined by the constraints of the maximum flow linear program LP1. A *vertex* of the polytope P is any feasible solution to LP1 with at least n_d tight linearly independent constraints. We will ensure that at every step of Ford-Fulkerson our flow f is a vertex of P . To do this we will make sure that the d -simplices corresponding to non-tight constraints of LP1 form an acyclic complex. Some straightforward algebra implies that this condition is enough to make at least n_d constraints tight. Let \mathcal{H}_f be the subcomplex of d -simplices “half-saturated” by f ; that is, $\sigma \in \mathcal{H}_f$ if and only if its capacity constraint is a strict inequality: $0 < f(\sigma) < c(\sigma)$. The half-saturated simplices do not make either of their two corresponding constraints tight, while d -simplices not in \mathcal{H}_f make exactly one of their corresponding constraints tight. We require that \mathcal{H}_f be an acyclic complex at each step of Ford-Fulkerson. In the case of graphs, this just means that \mathcal{H}_f is a forest. For a d -dimensional complex it means that $\dim H_d(\mathcal{H}_f) = 0$. Acyclic complexes have been studied by Duval, Klivans, and Martin who show that they share many properties with forests and trees in graphs [8]. The following lemma shows that if \mathcal{H}_f is acyclic then f is a vertex of the flow polytope.

► **Lemma 22.** *Let f be a feasible flow for the d -dimensional simplicial flow network (\mathcal{K}, c, γ) . If the subcomplex of half-saturated d -simplices \mathcal{H}_f is acyclic then f is a vertex of the flow polytope P .*

At each iteration of Ford-Fulkerson we want to pick an augmenting chain such that the resulting flow leaves \mathcal{H}_f acyclic. It’s not clear how to pick such an augmenting chain. However, no matter what augmenting chain we pick we can always repair the flow in a way that the resulting flow leaves \mathcal{H}_f acyclic. To do so we compute a homology basis of \mathcal{H}_f and update the flow to make $\dim H_d(\mathcal{H}_f) = 0$.

► **Lemma 23.** *Let f be a feasible flow for the d -dimensional simplicial flow network (\mathcal{K}, c, γ) . If the subcomplex of half-saturated d -simplices \mathcal{H}_f is not acyclic then in $O(n^{\omega+1})$ time we can construct a new flow f' such that $\mathcal{H}_{f'}$ is acyclic and $\|f\| = \|f'\|$.*

To wrap up the section, we state our main theorem whose proof is immediate from Lemmas 22 and 23.

► **Theorem 24.** *Given a simplicial flow network (\mathcal{K}, c, γ) we can compute a maximum flow f by using the Ford-Fulkerson algorithm with the following heuristic: at every iteration pick an augmenting chain such that the subcomplex of half-saturated d -simplices \mathcal{H}_f is acyclic.*

References

- 1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., USA, 1993.
- 2 Glencora Borradaile and Philip Klein. An $O(n \log n)$ algorithm for maximum st -flow in a directed planar graph. *J. ACM*, 56(2), 2009. doi:10.1145/1502793.1502798.
- 3 Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Minimum cuts and shortest homologous cycles. In *Proc. 25th Ann. Symp. Comput. Geom.*, pages 377–385, 2009.
- 4 Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM Journal on Computing*, 41(6):1605–1634, 2012. doi:10.1137/090766863.
- 5 Tamal K. Dey, Anil N. Hirani, and Bala Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM J. Comput.*, 40(4):1026–1044, 2011. doi:10.1137/100800245.
- 6 Tamal K. Dey, Tao Hou, and Sayan Mandal. Computing minimal persistent cycles: Polynomial and hard cases. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete*

- Algorithms*, SODA '20, page 2587–2606, USA, 2020. Society for Industrial and Applied Mathematics.
- 7 Art M. Duval, Caroline J. Klivans, and Jeremy L. Martin. Cuts and flows of cell complexes. *Journal of Algebraic Combinatorics*, 41:969–999, 2015.
 - 8 Art M. Duval, Caroline J. Klivans, and Jeremy L. Martin. *Simplicial and cellular trees*, pages 713–752. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-24298-9_28.
 - 9 Jeff Erickson. *Algorithms*. <http://algorithms.wtf>, 2019.
 - 10 Jeff Erickson and Amir Nayyeri. Minimum cuts and shortest non-separating cycles via homology covers. In *Proc. 22nd Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 1166–1176, 2011.
 - 11 Arthur F. Veinott, Jr. and George B. Dantzig. Integral extreme points. *SIAM Review*, 10(3):371–372, July 1968. doi:10.1137/1010063.
 - 12 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.*, 16(6):1004–1004, 1987.
 - 13 Refael Hassin. Maximum flow in (s, t) planar networks. *Inf. Process. Lett.*, 13:107, 1981.
 - 14 Refael Hassin and Donald B. Johnson. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comput.*, 14(3):612–624, 1985.
 - 15 Sergei Ivanov (<https://mathoverflow.net/users/4354/sergei-ivanov>). computational complexity. MathOverflow. URL:<https://mathoverflow.net/q/118428> (version: 2013-01-09). arXiv:<https://mathoverflow.net/q/118428>.
 - 16 Oscar H Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982. doi:10.1016/0196-6774(82)90007-4.
 - 17 Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM J. Comput.*, 8:135–150, 1979.
 - 18 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proc. 43rd Ann. ACM Symp. Theory Comput.*, pages 313–322, 2011.
 - 19 Fabian Latorre. The maxflow problem and a generalization to simplicial complexes, 2012. arXiv:arXiv:1212.1406.
 - 20 Lek-Heng Lim. Hodge Laplacians on graphs. *SIAM Review*, 63(3):685–715, 2020.
 - 21 Sarah Morell, Ina Seidel, and Stefan Weltge. Minimum-cost integer circulations in given homology classes, 2020. arXiv:1911.10912.
 - 22 John Reif. Minimum s - t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM J. Comput.*, 12:71–81, 1983.
 - 23 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., USA, 1986.

Hypersuccinct Trees – New Universal Tree Source Codes for Optimal Compressed Tree Data Structures and Range Minima

J. Ian Munro 


University of Waterloo, Canada

Patrick K. Nicholson¹ 

Nokia Bell Labs, Dublin, Ireland

Louisa Seelbach Benkner 

Universität Siegen, Germany

Sebastian Wild 

University of Liverpool, UK

Abstract

We present a new universal source code for distributions of unlabeled binary and ordinal trees that achieves optimal compression to within lower order terms for all tree sources covered by existing universal codes. At the same time, it supports answering many navigational queries on the compressed representation in constant time on the word-RAM; this is not known to be possible for any existing tree compression method. The resulting data structures, “hypersuccinct trees”, hence combine the compression achieved by the best known universal codes with the operation support of the best succinct tree data structures.

We apply hypersuccinct trees to obtain a universal compressed data structure for range-minimum queries. It has constant query time and the optimal worst-case space usage of $2n + o(n)$ bits, but the space drops to $1.736n + o(n)$ bits on average for random permutations of n elements, and $2 \lg \binom{n}{r} + o(n)$ for arrays with r increasing runs, respectively. Both results are optimal; the former answers an open problem of Davoodi et al. (2014) and Golin et al. (2016).

Compared to prior work on succinct data structures, we do not have to tailor our data structure to specific applications; hypersuccinct trees automatically adapt to the trees at hand. We show that they simultaneously achieve the optimal space usage to within lower order terms for a wide range of distributions over tree shapes, including: binary search trees (BSTs) generated by insertions in random order / Cartesian trees of random arrays, random fringe-balanced BSTs, binary trees with a given number of binary/unary/leaf nodes, random binary tries generated from memoryless sources, full binary trees, unary paths, as well as uniformly chosen weight-balanced BSTs, AVL trees, and left-leaning red-black trees.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Data structures design and analysis

Keywords and phrases analysis of algorithms, universal source code, compressed trees, succinct data structure, succinct trees, tree covering, random binary search trees, range-minimum queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.70

Related Version *Extended Version*: <https://arxiv.org/abs/2104.13457>

Funding This work has been supported in part by the Canada Research Chairs Programme and an NSERC Discovery Grant, the DFG research project LO 748/10-2 (QUANT-KOMP), and the NeST (Network Sciences and Technologies) EEECS School initiative of University of Liverpool.

Acknowledgements We thank Conrado Martínez and Markus Lohrey for valuable discussions and feedback on earlier drafts of this paper, as well as our anonymous reviewers for their comments.

¹ Research carried out while P. Nicholson was at Nokia Bell Labs, Ireland.



1 Introduction

As space usage and memory access become the bottlenecks in computation, working directly on a compressed representation (“computing over compressed data”) has become a popular field. For text data, substantial progress over the last two decades culminated in compressed text indexing methods that had wide-reaching impact on applications and satisfy strong analytical guarantees. For structured data, the picture is much less developed and clear. In this paper, we develop the analog of entropy-compressed string indices for trees: a data structure that allows one to query a tree stored in compressed form, with optimal query times and space matching the best universal tree codes.

Computing over compressed data became possible by combining techniques from information theory, string compression, and data structures. The central object of study in (classical) information theory is that of a *source* of random strings, whose entropy rate is the fundamental limit for source coding. The ultimate goal in compressing such strings is a *universal code*, which achieves optimal compression (to within lower order terms) for *distributions* of strings from a large class of possible sources *without knowing the used source*.

A classic result in this area is that Lempel-Ziv methods are universal codes for finite-state sources, i.e., sources in which the next symbol’s distribution depends on the previous k emitted symbols (see, e.g., [4, § 13]). The same is true for methods based on the Burrows-Wheeler-transform [6] and for grammar-based compression [30]. The latter two results were only shown around 2000, marking a renewed interest in compression methods.

The year 2000 also saw breakthroughs in compressed text indexing, with the first compressed self-indices that can represent a string and support pattern matching queries using $O(nH_0)$ bits of space [23, 24] and $O(nH_k) + o(n \log |\Sigma|)$ bits of space [8] for H_k the k th order empirical entropy of the string (for $k \geq 0$); many improvements have since been obtained on space and query time; (see [37, 1] for surveys and [15] for lower bounds on redundancy; [35, 36] summarizes more recent trends). For strings, computing over compressed data has mainly been achieved.

In this article, we consider structure instead of strings; focusing on one of the simplest forms of structured data: unlabeled binary and ordinal trees. Unlike for strings, the information theory of structured data is still in its infancy. Random sources of binary trees have (to our knowledge) first been suggested and analyzed in 2009 [31]; a more complete formalization then appeared in [43], together with a first universal tree source code.

For trees, computational results predate information-theoretic developments. *Succinct data structures* date back to 1989 [28] and have their roots in storing trees space-efficiently while supporting fast queries. A succinct data structure is allowed to use $\lg U_n(1 + o(1))$ bits of space to represent one out of U_n possible objects of size n – corresponding to a uniform distribution over these objects. This has become a flourishing field, and several succinct data structures for ordinal or cardinal (including binary) trees supporting many operations are known [34]. Apart from the exceptions discussed below (in particular [29, 5]), these methods do not achieve any compression beyond $\lg U_n$ no matter what the input is.

At the other end of the spectrum, more recent representations for highly repetitive trees [2, 3, 12, 14, 17, 18] can realize exponential space savings over $\lg U_n$ in extreme cases, but recent lower bounds [39] imply that these methods cannot simultaneously achieve constant time² for queries; they are also not known to be succinct when the tree is not highly compressible.

² All running times assume the word-RAM model with word size $w = \Theta(\log n)$.

■ **Table 1** Navigational operations on succinct binary trees. (v denotes a node and i an integer).

<code>parent(v)</code>	the parent of v , same as <code>anc(v, 1)</code>
<code>degree(v)</code>	the number of children of v
<code>left_child(v)</code>	the left child of node v
<code>right_child(v)</code>	the right child of node v
<code>depth(v)</code>	the depth of v , i.e., the number of edges between the root and v
<code>anc(v, i)</code>	the ancestor of node v at depth <code>depth(v) - i</code>
<code>nbdesc(v)</code>	the number of descendants of v
<code>height(v)</code>	the height of the subtree rooted at node v
<code>LCA(v, u)</code>	the lowest common ancestor of nodes u and v
<code>leftmost_leaf(v)</code>	the leftmost leaf descendant of v
<code>rightmost_leaf(v)</code>	the rightmost leaf descendant of v
<code>level_leftmost(ℓ)</code>	the leftmost node on level ℓ
<code>level_rightmost(ℓ)</code>	the rightmost node on level ℓ
<code>level_pred(v)</code>	the node immediately to the left of v on the same level
<code>level_succ(v)</code>	the node immediately to the right of v on the same level
<code>node_rank$_X$(v)</code>	the position of v in the X -order, $X \in \{\text{PRE}, \text{POST}, \text{IN}\}$, i.e., in a preorder, postorder, or inorder traversal of the tree
<code>node_select$_X$(i)</code>	the i th node in the X -order, $X \in \{\text{PRE}, \text{POST}, \text{IN}\}$
<code>leaf_rank(v)</code>	the number of leaves before and including v in preorder
<code>leaf_select(i)</code>	the i th leaf in preorder

In this paper, we fill this gap between succinct trees and dictionary-compressed trees by presenting the first data structure for unlabeled binary trees that answers all queries supported in previous succinct data structures (cf. Table 1) in $O(1)$ time and simultaneously achieves optimal compression over the same tree sources as the best previously known universal tree codes. We also extend the tree-source concepts and our data structure to unlabeled ordinal trees. In contrast to previous succinct trees, we give a single, *universal* data structure, the *hypersuccinct trees*³, that does not need to be adapted to specific classes or distributions of trees.

Our hypersuccinct trees require only a minor modification of existing succinct tree data structures based on tree covering [20, 25, 7], (namely Huffman coding micro-tree types); the contribution of our work is the careful analysis of the information-theoretic properties of the tree-compression method, the “hypersuccinct code”, that underlies these data structures.

As a consequence of our results, we solve an open problem for succinct range-minimum queries (RMQ): Here the task is to construct a data structure from an array $A[1..n]$ of comparable items at preprocessing time that can answer subsequent queries without inspecting A again. The answer to the query $\text{RMQ}(i, j)$, for $1 \leq i \leq j \leq n$, is the index (in A) of the (leftmost) minimum in $A[i..j]$, i.e., $\text{RMQ}(i, j) = \arg \min_{i \leq k \leq j} A[k]$. We give a data structure that answers RMQ in constant time using the optimal expected space of $1.736n + o(n)$ bits when the array is a random permutation, (and $2n + o(n)$ in the worst case); previous work either had suboptimal space [5] or $\Omega(n)$ query time [21]. We obtain the same (optimal) space usage for storing a binary search tree (BST) built from insertions in random order (“random BSTs” hereafter). Finally, we show that the space usage of our RMQ data structure is also bounded by $2 \lg \binom{n}{r} + o(n)$ whenever A has r increasing runs, and that this is again best possible.

³ The name “hypersuccinct trees” is the escalation of the “ultrasuccinct trees” of [29].

Outline. The rest of our article is structured as follows: A comprehensive list of the contributions appears below in Section 2. Section 3 describes our compressed tree encoding. In Section 4, we illustrate the techniques for proving universality of our hypersuccinct code on two well-known types of binary-trees shape distributions – random BSTs and weight-balanced trees – and sketch the extensions necessary for the general results. In Section 5, we present our RMQ data structures. Finally, Section 6 concludes the paper with future directions.

Appendix A contains missing proofs for the RMQ data structures. An extended version of this article is available online (<https://arxiv.org/abs/2104.13457>); it contains formal definitions of tree sources and full proofs for all universality statements.

2 Results

In a binary tree, each node has a left and a right child, either of which can be empty (“null”). For a binary tree t we denote by $|t|$ the number of nodes in t . Unless stated otherwise, $n = |t|$. A binary tree source \mathcal{S} emits a tree t with a certain probability $\mathbb{P}_{\mathcal{S}}[t]$ (potentially $\mathbb{P}_{\mathcal{S}}[t] = 0$); we write $\mathbb{P}[t]$ if \mathcal{S} is clear from the context. $\lg(1/0)$ is taken to mean $+\infty$.

► **Theorem 1** (Hypersuccinct binary trees). *Let t be a binary tree over n nodes. The hypersuccinct representation of t supports all queries from Table 1 in $O(1)$ time and uses $|H(t)| + o(n)$ bits of space, where*

$$|H(t)| \leq \min \left\{ 2n + 1, \min_{\mathcal{S}} \lg \left(\frac{1}{\mathbb{P}_{\mathcal{S}}[t]} \right) + o(n) \right\},$$

and $\mathbb{P}_{\mathcal{S}}[t]$ is the probability that t is emitted by source \mathcal{S} . The minimum is taken over all binary-tree sources \mathcal{S} in the following families (which are explained in Table 4):

- (i) memoryless node-type processes,
- (ii) k th-order node-type processes (for $k = o(\log n)$),
- (iii) monotonic fixed-size sources,
- (iv) worst-case fringe-dominated fixed-size sources,
- (v) monotonic fixed-height sources,
- (vi) worst-case fringe-dominated fixed-height sources,
- (vii) tame uniform subclass sources.

► **Corollary 2** (Hypersuccinct binary trees: Examples & Empirical entropies). *Hypersuccinct trees achieve optimal compression to within lower order terms for all example distributions listed in Table 3. Moreover, for every binary tree t , we have:*

- (i) $|H(t)| \leq H_k^{\text{type}}(t) + o(n)$ with $H_k^{\text{type}}(t)$ the (unnormalized) k th-order empirical entropy of node types (leaf, left-unary, binary, or right-unary) for $k = o(\log n)$.
- (ii) $|H(t)| \leq H_{st}(t) + o(n)$ with $H_{st}(t)$ the “subtree-size entropy”, i.e., the sum of the logarithm of the subtree size of v for all nodes v in t , (a.k.a. the splay-tree potential).

The hypersuccinct code is a *universal code* for the families of binary-tree sources listed in Theorem 1 with bounded maximal pointwise redundancy. We also present a more general class of sources, for which our code achieves $o(n)$ expected redundancy in the appendix; see also Table 4.

To our knowledge, the list in Theorem 1 is a comprehensive account of *all* concrete binary-tree sources for which any universal code is known. Remarkably, in all cases the bounds on redundancies proven for the hypersuccinct code are identical (up to constant factors) to those known for existing universal binary-tree codes. *Our hypersuccinct code thus achieves the same compression as all previous universal codes, but simultaneously supports constant-time queries on the compressed representation with $o(n)$ overhead.*

■ **Table 2** Overview of random tree sources for binary and ordinal trees.

Name	Notation	Intuition	Formal Definition of $\mathbb{P}[t]$
Memoryless Processes	τ	A binary tree is constructed top-down, drawing each node's <i>type</i> (0 = leaf, 1 = left-unary, 2 = binary, 3 = right-unary) i.i.d. according to the distribution $(\tau_0, \tau_1, \tau_2, \tau_3)$.	$\mathbb{P}[t] = \prod_{v \in t} \tau(\text{type}(v))$
Higher-order Processes	$(\tau_z)_z$	A binary tree is constructed top-down, drawing node v 's <i>type</i> according to $\tau_{h_k(v)} : \{0, 1, 2, 3\} \rightarrow [0, 1]$, which depends on the types of the k closest <i>ancestors</i> of v .	$\mathbb{P}[t] = \prod_{v \in t} \tau_{h_k(v)}(\text{type}(v))$
Fixed-size Binary Tree Sources	$\mathcal{S}_{fs}(p)$	A binary tree of size n is constructed top-down, asking source p at each node for its left- and right <i>subtree size</i> .	$\mathbb{P}[t] = \prod_{v \in t} p(t_\ell(v) , t_r(v))$ $t_{\ell/r}(v)$ = left/right subtree of v
Fixed-height Binary Tree Sources	$\mathcal{S}_{fh}(p)$	A binary tree of height h is constructed top-down, asking source p at each node for a left and right subtree <i>height</i> .	$\mathbb{P}[t] = \prod_{v \in t} p(h(t_\ell(v)), h(t_r(v)))$ $h(t)$ = height of t
Uniform Subclass Sources	$\mathcal{U}_{\mathcal{P}}$	A binary tree is drawn uniformly at random from the set $\mathcal{T}_n(\mathcal{P})$ of all binary trees of size n that satisfy property \mathcal{P} .	$\mathbb{P}[t] = \frac{1}{ \mathcal{T}_n(\mathcal{P}) }$
Memoryless Ordinal Tree Sources	d	An ordinal tree is constructed top-down, drawing each node v 's degree $\deg(v)$ according to distribution $d = (d_0, d_1, \dots)$.	$\mathbb{P}[t] = \prod_{v \in t} d_{\deg(v)}$
Fixed-size Ordinal Tree Sources	$\mathcal{S}_{fs}(p)$	An ordinal tree of size n is constructed top-down, asking source p at each node for the number and sizes of the subtrees.	$\mathbb{P}[t] = \prod_{v \in t} p(t_1[v] , \dots, t_{\deg(v)}[v])$

In terms of queries, previous solutions either have suboptimal query times [2, 3, 14], higher space usage [39], or rely on tailoring the representation to a specific subclass of trees [29, 7] to achieve good space and time for precisely these instances, but they fail to generalize to other use cases. Some also do not support all queries. We give a detailed comparison with the state of the art in the extended version of this article.

We focus here on our results for binary trees. In the extended version of this article, we extend the above notions of tree sources (except fixed-height sources) to ordinal trees, which has not been done to our knowledge. Moreover, we extend both our code and data structure to ordinal trees, and show their universality for these sources.

3 From Tree Covering to Hypersuccinct Trees

Our universally compressed tree data structures are based on *tree covering* [20, 25, 7]: A (binary or ordinal) tree t is decomposed into *mini trees*, each of which is further decomposed into *micro trees*; the size of the latter, $B = B(n) = \lg n/8$, is chosen so that we can tabulate all possible shapes of micro trees and the answers to various micro-tree-local queries in one global lookup table (the “Four-Russian Table” technique). For each micro tree, its local shape is stored, e.g., using the balanced-parenthesis (BP) encoding, using a total of exactly $2n$ bits (independent of the tree shape). Using additional data structures occupying only $o(n)$ bits of space, a long list of operations can be supported in constant time (Table 1). The space usage of this representation is optimal to within lower order terms for the worst case, since $\lg C_n \sim 2n$ bits are necessary to distinguish all $C_n = \binom{2n}{n}/(n+1)$ trees of n nodes. (This worst-case bound applies both to ordinal trees and binary trees).

■ **Table 3** An overview over the concrete examples of tree-shape distributions that our hypersuccinct code compresses optimally (up to lower-order terms).

Tree-Shape Distribution	Entropy	Corresponding Source
(Uniformly random) binary trees of size n	$2n$	Memoryless binary, monotonic fixed-size binary
(Uniformly random) full binary trees of size n	n	Memoryless binary
(Uniformly random) unary paths of length n	n	Memoryless binary
(Uniformly random) Motzkin trees of size n	$1.585n$	Memoryless binary
Binary search trees generated by insertions in random order (“random BSTs”)	$1.736n$	Monotonic fixed-size binary
Binomial random trees	$P(\lg n)n^a)$	Average-case fringe-dominated fixed-size binary
Almost paths	— ^{b)}	Monotonic fixed-size binary
Random fringe-balanced binary search trees	— ^{b)}	Average-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of height h	— ^{b)}	Worst-case fringe-dominated fixed-height binary
(Uniformly random) weight-balanced binary trees of size n	— ^{b)}	Worst-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of size n	$0.938n$	Uniform-subclass
(Uniformly random) left-leaning red-black trees of size n	$0.879n$	Uniform-subclass
(Uniformly random) full m -ary trees of size n	$\lg(\frac{m}{m-1})n$	Memoryless ordinal
Uniform composition trees	— ^{b)}	Monotonic fixed-size ordinal
Random LRM-trees	$1.736n$	Monotonic fixed-size ordinal

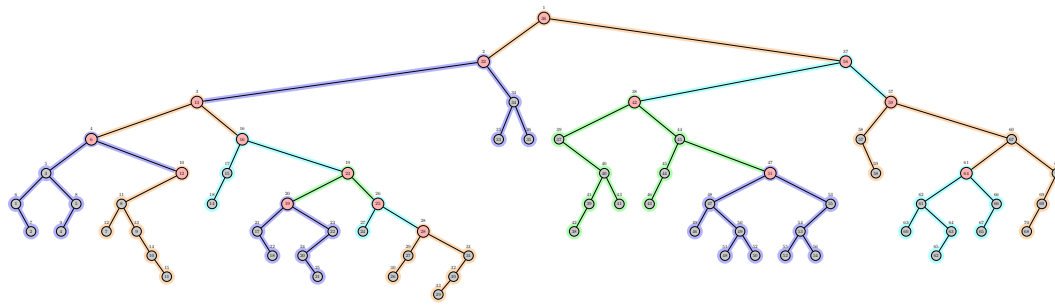
a) Here P is a nonconstant, continuous, periodic function with period 1.

b) No (concise) asymptotic approximation known.

■ **Table 4** Sufficient conditions under which we show universality of our hypersuccinct code H for binary trees. Proofs are given in the extended version of this article (<https://arxiv.org/abs/2104.13457>).

Family of sources	Restriction	Redundancy
Memoryless node-type	—	$O(n \log \log n / \log n)$
k th-order node-type	—	$O((nk + n \log \log n) / \log n)$
Monotonic fixed-size	$p(\ell, r) \geq p(\ell + 1, r)$ and $p(\ell, r) \geq p(\ell, r + 1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log \log n / \log n)$
Worst-case fringe-dominated fixed-size	$n_{\geq B}(t) = o(n / \log \log n)$ for all t with $\mathbb{P}[t] > 0$; $n_{\geq B}(t) = \#\text{nodes with subtree size in } \Omega(\log n)$	$O(n_{\geq B}(t) \log \log n + n \log \log n / \log n)$
Weight-balanced fixed-size	$\sum_{\frac{n}{c} \leq \ell \leq n - \frac{n}{c}} p(\ell - 1, n - \ell - 1) = 1$ for constant $c \geq 3$	$O(n \log \log n / \log n)$
Average-case fringe-dominated fixed-size	$\mathbb{E}[n_{\geq B}(T)] = o(n / \log \log n)$ for random T generated by source \mathcal{S}	$O(n_{\geq B}(t) \log \log n + n \log \log n / \log n)^a)$
Monotonic fixed-height	$p(\ell, r) \geq p(\ell + 1, r)$ and $p(\ell, r) \geq p(\ell, r + 1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log \log n / \log n)$
Worst-case fringe-dominated fixed-height	$n_{\geq B}(t) = o(n / \log \log n)$ for all t with $\mathbb{P}[t] > 0$	$O(n_{\geq B}(t) \log \log n + n \log \log n / \log n)$
Tame uniform-subclass	class of trees $\mathcal{T}_n(\mathcal{P})$ is hereditary (i.e., closed under taking subtrees), $n_{\geq B}(t) = o(n / \log \log n)$ for $t \in \mathcal{T}_n(\mathcal{P})$, $\lg \mathcal{T}_n(\mathcal{P}) = cn + o(n)$ for constant $c > 0$, heavy-twigged: if v has subtree size $\Omega(\log n)$, v 's subtrees have size $\omega(1)$	$o(n)$

a) Stated redundancy is achieved in expectation for a random tree t generated by the source.



■ **Figure 1** Example binary tree with $n = 70$ nodes and micro trees computed by the Farzan-Munro tree-covering algorithm [7] with parameter $B = 6$. The micro trees are indicated by colors. The algorithm guarantees that each node is part of exactly one micro tree and that each micro tree has at most three edges shared with other micro trees, namely to a parent, a left- and a right-child micro tree.

A core observation is that the dominant space in tree-covering data structures comes from storing the micro-tree types, and these can be further compressed using a different code. This has been used in an ad-hoc manner for specific tree classes [7, 5, 16], but has not been investigated systematically. A natural idea is to use a Huffman code for the micro tree types to simultaneously beat the compression of all these special cases; we dub this as the “Four Russians and One American”⁴ trick. Applying it to the data structures based on the Farzan-Munro tree-covering algorithm [7] yields our hypersuccinct trees.

The main contribution of our present work is the careful analysis of the potential of the Four Russians and One American trick for (binary and ordinal) tree source coding. As an immediate corollary, we obtain a single data structure that achieves optimal compression for all special cases covered in previous work, plus a much wider class of distributions over trees for which no efficient data structure was previously known.

Our analysis builds on previous work on tree compression, specifically DAG compression and tree straight-line programs (TSLPs) [32]. Our core idea is to interpret (parts of the) tree-covering data structures as a *code* for trees, the “hypersuccinct code”: it stores the type, i.e., the local shape, of all micro trees separately from how they interface to form the entire tree. Intuitively, our hypersuccinct code is a restricted version of a grammar-based tree code, where we enforce having nonterminals for certain subtrees;⁵ we strengthen and extend existing universality proofs from general grammar-based tree codes to the restricted hypersuccinct code.

4 Universality for Fixed-Size Sources

In this section, we sketch the proof that our hypersuccinct trees achieve optimal compression for two exemplary tree-shape distributions: random binary search trees and uniform weight-balanced trees (defined below). These examples serve to illustrate the proof techniques and

⁴ It deems us only fair to do D. A. Huffman the same questionable honor of reducing the person to a country of residence that V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev have experienced ever since their table-lookup technique has become known as the “Four-Russians trick”.

⁵ Differences in technical details make the direct comparison difficult, though: in TSLPs, holes in contexts must be stored (and encoded) alongside the local shapes as they are both part of the right-hand side of productions; in our hypersuccinct code, we separately encode the shapes of micro trees and the positions of portals, potentially gaining a small advantage. Our comment thus remains a motivational hint as to why similar analysis techniques are useful in both cases, but falls short of providing a formal reduction.

to showcase the versatility of the approach. The extension to the general sufficient conditions from Table 4 and full details of computations are spelled out in the extended version of the article.

By *random BSTs*, we mean the distribution of tree shapes obtained by successively inserting n keys in random order into an (initially empty) unbalanced binary search tree (BST). We obtain random BSTs from a fixed-size tree source $\mathcal{S}_{fs}(p_{bst})$ with $p_{bst}(\ell, n-1-\ell) = \frac{1}{n}$ for all $\ell \in \{0, \dots, n-1\}$ and $n \in \mathbb{N}_{\geq 1}$, i.e., making every possible split equally likely. (Any left subtree size ℓ is equally likely in a random BST of a given size n .) Hence, $\mathbb{P}[t] = \prod_{v \in t} 1/|t[v]|$ where $t[v]$ is the subtree rooted at v and $|t[v]|$ its size (in number of nodes).

The second example are the shapes of uniformly random *weight-balanced BSTs* (BB[α]-trees, [38]): A binary tree t is α -weight-balanced if we have for every node v in t that $\min\{|t_\ell[v]|, |t_r[v]|\} + 1 \geq \alpha(|t[v]| + 1)$. Here $t_\ell[v]$ resp. $t_r[v]$ are the left resp. right subtrees of $t[v]$. We denote the set of α -weight-balanced trees of size n by $\mathcal{T}_n(\mathcal{W}_\alpha)$. We obtain random α -weight-balanced trees from another fixed-size source $\mathcal{S}_{fs}(p_{wb})$ with

$$p_{wb}(\ell, n-1-\ell) = \begin{cases} \frac{|\mathcal{T}_\ell(\mathcal{W}_\alpha)| |\mathcal{T}_{n-1-\ell}(\mathcal{W}_\alpha)|}{|\mathcal{T}_n(\mathcal{W}_\alpha)|} & \text{if } \min\{\ell+1, n-\ell\} \geq \alpha(n+1), \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that this yields the uniform probability distribution on $\mathcal{T}_n(\mathcal{W}_\alpha)$, i.e., with $\mathbb{P}[t] = 1/|\mathcal{T}_n(\mathcal{W}_\alpha)|$ for $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$ and $\mathbb{P}[t] = 0$ otherwise. We note that computing $|\mathcal{T}_n(\mathcal{W}_\alpha)|$ is a formidable challenge in combinatorics, but we never have to do so; we only require the *existence* of the fixed-size source for weight-balanced BSTs.

The *hypersuccinct code* $H(t)$ is formed by partitioning the nodes of a given binary tree t into $m = \Theta(n/\log n)$ micro trees μ_1, \dots, μ_m , each of which is a connected subtree of at most $\mu = O(\log n)$ nodes; an example is shown in Figure 1. Previous work on tree covering shows how to compute these and how to encode everything but the local shape of the micro trees in $o(n)$ bits of space [7]. (For a mere encoding, $O(n \log \log n / \log n)$ bits suffice.

The dominant part of the hypersuccinct code is the list of *types* of all micro trees, i.e., the (local) shapes of the induced subtrees formed by the set of nodes in the micro trees. Let C be a Huffman code for the string μ_1, \dots, μ_m , where we identify micro trees with their types. For a variety of different tree sources \mathcal{S} , we can prove that $\sum_{i=1}^m |C(\mu_i)|$, the total length of codewords for the micro trees, is upper bounded by $\lg(1/\mathbb{P}[t]) + \text{lower-order terms}$, where $\mathbb{P}[t]$ is the probability that t is emitted by \mathcal{S} ; this is the best possible code length to within lower order terms achievable for that source. We will now show this for our two example distributions.

4.1 Random BSTs

The proof consists of four steps that can be summarized as follows:

<p>Step 1 Construct a source-specific micro-tree encoding $D_S: \{\mu_1, \dots, \mu_m\} \rightarrow \{0, 1\}^*$ Goal: $D_S(\mu_i) \approx \lg(1/\mathbb{P}[\mu_i])$</p>	<p>Step 2 By optimality of Huffman codes: $\sum_{i=1}^m C(\mu_i) \leq \sum_{i=1}^m D_S(\mu_i)$</p>	<p>Step 3 Use properties of \mathcal{S} to show that $\prod_{i=1}^m \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$</p>	<p>Step 4 Conclude $\sum_{i=1}^m C(\mu_i) \approx \lg(1/\mathbb{P}[t])$</p>
---	--	---	---

Steps 2 and 4 do not depend on the source and indeed follow immediately; Steps 1 and 3 are the creative parts. Ignoring proper tracing of error terms, the result then follows as

$$|H(t)| \sim \sum_{i=1}^n |C(\mu_i)| \leq \sum_{i=1}^n |D_{\mathcal{S}}(\mu_i)| \leq \sum_{i=1}^n \lg(1/\mathbb{P}[\mu_i]) \lesssim \lg(1/\mathbb{P}[t]).$$

Let us consider $\mathcal{S}_{fs}(p_{bst})$, the fixed-size source producing (shapes of) random BSTs, and address these steps independently.

Our task in Step 1 is to find a code $D_{\mathcal{S}}$ for the micro-tree types that can occur in t , so that $|D_{\mathcal{S}}(\mu_i)| = \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + O(\log \log n)$. This code may rely on the decoder to have knowledge of \mathcal{S} . For random BSTs, $D_{\mathcal{S}}(t)$ can be constructed as follows: We initially store n using Elias gamma code⁶ and then, following a depth-first (preorder) traversal of the tree, we encode the size of the left subtree using *arithmetic coding*. Inductively, the size of the currently encoded node is always known, and the source-specific code is allowed to use the probability distributions hardwired into \mathcal{S} without storing them; for random BSTs, we simply encode a number uniformly distributed in $[0..s-1]$ at a node with subtree size s , using exactly $\lg s$ bits. Apart from storing the initial size and the small additive overhead from arithmetic coding, the code length of this “depth-first arithmetic tree code” is best possible: $|D_{\mathcal{S}}(t)| \leq \lg(1/\mathbb{P}[t]) + O(\log |t|)$. This concludes Step 1.

For Step 3, we have to show that the probability for the entire tree t is at most the product of the probabilities for all micro-trees. Recall that μ_1, \dots, μ_m are the micro trees in t . We can write $\mathbb{P}[t]$ as a product over contributions of individual nodes, and can collect factors in $\mathbb{P}[t]$ according to micro trees; this works for any fixed-size source. For random BSTs, we can use the “monotonicity” of node contributions to show

$$\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|} = \prod_{i=1}^m \prod_{v \in \mu_i} \frac{1}{|t[v]|} \leq \prod_{i=1}^m \prod_{v \in \mu_i} \frac{1}{|\mu_i[v]|} = \prod_{i=1}^m \mathbb{P}[\mu_i].$$

That completes Step 3, and hence the proof that $|H(t)| \leq \mathbb{P}_{\mathcal{S}_{fs}(p_{bst})}[t] + o(n)$.

4.2 Weight-balanced trees

Let us now consider uniformly random weight-balanced trees, i.e., the source $\mathcal{S} = \mathcal{S}_{fs}(p_{wb})$. We would like to follow the same template as above; however, this is not possible: Step 3 from above is in general not true anymore. The reason is that it is not clear whether the “non-fringe” micro trees, i.e., those that do not contain all descendants of the micro-tree root, have non-zero probability under \mathcal{S} . (A subtree of a tree is called *fringe*, if it consists of a node and all its descendants). Such micro trees will also make Step 1 impossible as they would require a code length of 0. While this issue is inevitable in general (Remark 4), we can under certain conditions circumvent Steps 1 and 3 altogether by directly bounding $\sum_{i=1}^m |D_{\mathcal{S}}(\mu_i)| \leq \mathbb{P}[t] + o(n)$.

As a first observation, note that it suffices to have $|D_{\mathcal{S}}(\mu_i)| = \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + O(\log \log n)$ for *all but a vanishing fraction* of the micro trees in any tree t ; then we can still hope to show $\sum_{i=1}^m |D_{\mathcal{S}}(\mu_i)| \leq \mathbb{P}[t] + o(n)$ overall. Second, it is known [12] that weight-balanced trees are “*fringe dominated*” in the following sense: Denoting by $n_{\geq B}(t)$ the number of “heavy” nodes, i.e., v in t with $|t[v]| \geq B = \lg n/8$, we have $n_{\geq B}(t) = O(n/B) = o(n)$ for

⁶ Elias gamma code $\gamma : \mathbb{N} \rightarrow \{0,1\}^*$ encodes an integer $n \geq 1$ using $2\lfloor \lg n \rfloor + 1$ bits by prefixing the binary representation of n with that representation’s length encoded in unary.

every weight-balanced tree $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$. Since only a vanishing fraction of nodes are heavy, one might hope that also only a vanishing fraction of micro trees are non-fringe, making the above route succeed. Unfortunately, that is not the case; the non-fringe micro trees can be a constant fraction of all micro trees.

Notwithstanding this issue, a more sophisticated micro-tree code D_S allows us to proceed. D_S encodes any *fringe* micro tree using a depth-first arithmetic code as for random BSTs. Any non-fringe micro tree μ_i , however, is broken up into the subtree of heavy nodes, the “boughs” of μ_i , and (fringe) subtrees $f_{i,j}$ hanging off the boughs. It is a property of the Farzan-Munro algorithm that every micro-tree root is heavy, hence all $f_{i,j}$ are indeed entirely contained within μ_i .

$D_S(\mu_i)$ then first encodes the bough nodes using 2 bits per node (using a BP representation for the boughs subtree) and then appends the depth-first arithmetic code for the $f_{i,j}$ (in left-to-right order). While this does not actually achieve $|D_S(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$ for entire micro trees μ_i , it does so for all the fringe subtrees $f_{i,j}$. Any node not contained in a fringe subtree $f_{i,j}$ must be part of a bough and hence heavy; by the fringe-dominance property, these nodes form a vanishing fraction of all nodes and hence contribute $o(n)$ bits overall. This shows that $|\mathbf{H}(t)| \leq \mathbb{P}_{\mathcal{S}_{fs}(\mathcal{P}_{ub})}[t] + o(n)$.

► **Remark 3 (A simple code whose analysis isn’t).** It is worth pointing out that the source specific code D_S is only a vehicle for the *analysis* of $|\mathbf{H}(t)|$; the complicated encodings D_S do not ever need to be computed when using our codes or data structures.

4.3 Other Sources

For memoryless sources, the analysis follows the four-step template, and is indeed easier than the random BSTs since Step 3 becomes trivial. For higher-order sources, in order to know the node types of the k closest ancestors (in t) of all nodes of depth $\leq k$ in μ_i , we prefix the depth-first arithmetic code by the node types of the k closest ancestors of the root of μ_i . Then the k ancestor types are known inductively for all nodes in a preorder traversal of μ_i .

The tame uniform-subclass sources require the most technical proof, but it is conceptually similar to the weight-balanced trees from above. The source-specific encoding for fringe subtrees is trivial here: we can simply use the rank in an enumeration of all trees of a given size, prefixed by the size of the subtree. Using the tameness conditions, one can show that a similar decomposition into boughs and fringe subtrees yields an optimal code length for almost all nodes. Details are deferred to the extended version of this article.

* * *

Together with the observations from Section 3 this yields Theorem 1. We obtained similar results for ordinal trees; details are deferred to the extended version of this article.

► **Remark 4 (Restrictions are inevitable).** We point out that some restrictions like the ones discussed above cannot possibly be overcome in general. Zhang, Yang, and Kieffer [43] prove that the unrestricted class of fixed-size sources (leaf-centric binary tree sources in their terminology) does not allow a universal code, even when only considering expected redundancy. The same is true for unrestricted fixed-height and uniform-subclass sources. While each is a natural formalism to describe possible binary-tree sources, additional conditions are strictly necessary for any interesting compression statements to be made. Our sufficient conditions are the weakest such restrictions for which any universal source code is known to exist ([43, 13, 40]), even without the requirement of efficient queries.

5 Hypersuccinct Range-Minimum Queries

We now show how hypersuccinct trees imply an optimal-space solution for the range-minimum query (RMQ) problem.⁷ Let $A[1..n]$ store the numbers x_1, \dots, x_n , i.e., x_j is stored at index j for $1 \leq j \leq n$. While duplicates naturally arise in some applications, e.g., in the longest-common extension (LCE) problem, we assume here that x_1, \dots, x_n are n distinct numbers to simplify the presentation. However, our RMQ solution works regardless of which minimum-value index is to be returned so long as the tie breaking rule is deterministic and fixed at construction time.

5.1 Cartesian Trees

The *Cartesian tree* T for x_1, \dots, x_n (resp. for $A[1..n]$) is a binary tree defined recursively as follows: If $n = 0$, it is the empty tree (“null”). Otherwise it consists of a root whose left child is the Cartesian tree for x_1, \dots, x_{j-1} and its right child is the Cartesian tree for x_{j+1}, \dots, x_n where j is the position of the minimum, $j = \arg \min_k A[k]$. A classic observation of Gabow et al. [11] is that range-minimum queries on A are equivalent to lowest-common-ancestor (LCA) queries on T when identifying nodes with their inorder rank:

$$\text{RMQ}_A(i, j) = \text{node_rank}_{\text{IN}}\left(\text{LCA}(\text{node_select}_{\text{IN}}(i), \text{node_select}_{\text{IN}}(j))\right).$$

We can thus reduce an RMQ instance (on an arbitrary input) to an LCA instance on binary trees of the same size; (the number of nodes in T equals the length of the array).

5.2 Random RMQ

We first consider the random permutation model for RMQ: Every (relative) ordering of the elements in $A[1..n]$ is equally likely. Without loss of generality, we identify the n elements with their ranks, i.e., $A[1..n]$ contains a random permutation of $[1..n]$. We refer to this as a random RMQ instance.

We can characterize the distribution of the Cartesian tree associated with such a random RMQ instance: Since the minimum in a random permutation is located at every position $i \in [n]$ with probability $\frac{1}{n}$, the inorder index of the root is uniformly distributed in $[n]$. Apart from renaming, the subarrays $A[1..i-1]$ (resp. $A[i+1..n]$) contain a random permutation of $i-1$ (resp. $n-i$) elements, and these two permutations are independent of each other conditional on their sizes. Cartesian trees of random RMQ instances thus have the same distribution as random BSTs, and in particular shape t arises with probability $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|}$. The former are also known as random increasing binary trees [10, Ex. II.17 & Ex. III.33].

Since the sets of answers to range-minimum queries is in bijection with Cartesian trees, the entropy H_n of the distribution of the shape of the Cartesian tree (and hence random BSTs) gives an information-theoretic lower bound for the space required by any RMQ data structure (in the encoding model studied here). Kieffer, Yang and Szpankowski [31] show⁸

⁷ A technical report containing preliminary results for random RMQ, but including more details on the data structure aspects of our solution, can be found on arXiv [33].

⁸ Hwang and Neininger [26] showed already in 2002 that the quicksort recurrence can be solved explicitly for arbitrary toll functions. H_n satisfies this recurrence with toll function $\lg n$, hence they implicitly proved Equation (1).

that the entropy of random BSTs $H_n = \mathbb{E}_T[\lg(1/\mathbb{P}[T])] = \mathbb{E}_T[\sum_{v \in T} \lg(|T[v]|)]$ is

$$H_n = \lg(n) + 2(n+1) \sum_{i=2}^{n-1} \frac{\lg i}{(i+2)(i+1)} \sim 2n \sum_{i=2}^{\infty} \frac{\lg i}{(i+2)(i+1)} \approx 1.7363771n. \quad (1)$$

With these preparations, we are ready to prove our first result on range-minimum queries.

► **Corollary 5** (Average-case optimal succinct RMQ). *There is a data structure that supports (static) range-minimum queries on an array A of n (distinct) numbers in $O(1)$ worst-case time and which occupies $H_n + o(n) \approx 1.736n + o(n)$ bits of space on average over all possible permutations of the elements in A . The worst case space usage is $2n + o(n)$ bits.*

Proof. We construct a hypersuccinct tree on the Cartesian tree for A . It supports `node_rankIN`, `node_selectIN`, and `LCA` in $O(1)$ time and thus `RMQ` in constant time without access to A . By Corollary 2, the space usage of hypersuccinct trees is at most $\min\{2n, \lg(1/\mathbb{P}[t])\} + o(n)$ for $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|}$. By the above observations, this is the probability to obtain t as the Cartesian trees of a random permutation, so we store t with maximal pointwise redundancy of $o(n)$, hence also $o(n)$ expected redundancy over the entropy $H_n \sim 1.736n$. ◀

5.3 RMQ with Runs

A second example of compressible RMQ instances results from partially sorted arrays. Suppose that $A[1..n]$ can be split into r runs, i.e., maximal contiguous ranges $[j_i, j_{i+1} - 1]$, ($i = 1, \dots, r$ with $j_1 = 1$ and $j_{r+1} = n + 1$), so that $A[j_i] \leq A[j_i + 1] \leq \dots \leq A[j_{i+1} - 1]$.

► **Theorem 6** (Lower bound for RMQ with runs). *Any range-minimum data structure in the encoding model for an array of length n that contains r runs must occupy at least $\lg N_{n,r} \geq 2 \lg \binom{n}{r} - O(\log n)$ bits of space where $N_{n,r} = \frac{1}{n} \binom{n}{r} \binom{n}{r-1}$ are the Narayana numbers.*

The proof follows from a bijection between Cartesian trees on sequences of length n with exactly r runs and mountain-valley diagrams (a.k.a. Dyck paths) of length $2n$ with exactly r “peaks”; the latter is known to be counted by the *Narayana numbers* [27]. Details are given in Appendix A.

► **Corollary 7** (Optimal succinct RMQ with runs). *There is a data structure that supports (static) range-minimum queries on an array A of n numbers that consists of r runs in $O(1)$ worst-case time and which occupies $2 \lg \binom{n}{r} + o(n) \leq 2n + o(n)$ bits of space.*

This follows from the observation that a node’s type in the Cartesian tree, i.e., whether or not its left resp. right child is empty, closely reflects the runs in A : A binary node marks the beginning of a non-singleton run, a leaf node marks the last position in a non-singleton run, a right-unary node (i.e., left child empty, right child nonempty) is a middle node of a run, and a left-unary node corresponds to a singleton run. With $s \in [0..r]$ the number of singleton runs, we can bound the space for a hypersuccinct tree in terms of its empirical node-type entropy by $H_0^{\text{type}}(T) + o(n) = nH\left(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n}\right) + o(n)$, which can be shown to be no more than $2 \lg \binom{n}{r} + o(n)$ for any value of s ; again, details are deferred to Section A.

* * *

We close by pointing out that hypersuccinct trees simultaneously achieve the optimal bounds for RMQ on random permutations and arrays with r runs without taking explicit precautions for either. The same is true for any other shape distributions of Cartesian trees that can be written as one of the sources from Table 4.

6 Conclusion

We presented the first succinct tree data structures with optimally adaptive space usage for a large variety of random tree sources, both for binary trees and for ordinal trees. This is an important step towards the goal of efficient computation over compressed structures, and has immediate applications, e.g., as illustrated above for the range-minimum problem.

A goal for future work is to reduce the redundancy of $o(n)$, which becomes dominant for sources with sublinear entropy. While this has been considered for tree covering in principle [41], many details remain to be thoroughly investigated.

For very compressible trees, the space savings in hypersuccinct trees are no longer competitive. On the other hand, with current methods for random access on dictionary-compressed sequences, constant-time queries are not possible in the regime of mildly compressible strings; the same applies to known approaches to represent trees. An interesting question is whether these opposing approaches can be combined in a way to complement each other's strengths. We leave this direction for future work.

References

- 1 Djamal Belazzougui, Veli Mäkinen, and Daniel Valenzuela. Compressed suffix array. In *Encyclopedia of Algorithms*, pages 1–6. Springer US, 2014. doi:10.1007/978-3-642-27848-8_82-2.
- 2 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, 2015. doi:10.1016/j.ic.2014.12.012.
- 3 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, January 2015. doi:10.1137/130936889.
- 4 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Interscience, 2nd edition, 2006.
- 5 Pooya Davoodi, Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Encoding range minima and range top-2 queries. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130131–20130131, April 2014. doi:10.1098/rsta.2013.0131.
- 6 M. Effros, K. Visweswariah, S. R. Kulkarni, and S. Verdu. Universal lossless source coding with the burrows wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, May 2002. doi:10.1109/18.995542.
- 7 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014. doi:10.1007/s00453-012-9664-0.
- 8 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Comput. Soc, 2000. doi:10.1109/sfcs.2000.892127.
- 9 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, January 2011. doi:10.1137/090779759.
- 10 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. (available on author's website: <http://algo.inria.fr/flajolet/Publications/book.pdf>).
- 11 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *STOC 1984*. ACM Press, 1984. doi:10.1145/800057.808675.
- 12 Moses Ganardi, Danny Hucce, Artur Jez, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, June 2017. doi:10.1016/j.jcss.2016.12.007.

- 13 Moses Ganardi, Danny Hucce, Markus Lohrey, and Louisa Seelbach Benkner. Universal tree source coding using grammar-based compression. *IEEE Transactions on Information Theory*, 65(10):6399–6413, October 2019. doi:10.1109/tit.2019.2919829.
- 14 Moses Ganardi, Danny Hucce, Markus Lohrey, and Eric Noeth. Tree compression using string grammars. *Algorithmica*, 80(3):885–917, 2017. doi:10.1007/s00453-017-0279-3.
- 15 Michał Ganczorz. Entropy lower bounds for dictionary compression. 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 11:1–11:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.11.
- 16 Michał Gańczorz. Using statistical encoding to achieve tree succinctness never seen before. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, *LIPIcs*, pages 22:1–22:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.22.
- 17 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-based compression of unranked trees. *Theory of Computing Systems*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.
- 18 Paweł Gawrychowski and Artur Jez. LZ77 Factorisation of Trees. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65 of *LIPIcs*, pages 35:1–35:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2016.35.
- 19 Paweł Gawrychowski and Patrick K. Nicholson. Optimal encodings for range top- k , selection, and min-max. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 593–604, 2015. doi:10.1007/978-3-662-47672-7_48.
- 20 Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, October 2006. doi:10.1145/1198513.1198516.
- 21 Mordecai Golin, John Iacono, Danny Krizanc, Rajeev Raman, Srinivasa Rao Satti, and Sunil Shende. Encoding 2d range maximum queries. *Theoretical Computer Science*, 609:316–327, January 2016. doi:10.1016/j.tcs.2015.10.012.
- 22 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation For Computer Science*. Addison-Wesley, 1994.
- 23 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2000. doi:10.1145/335305.335351.
- 24 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, January 2005. doi:10.1137/s0097539702402354.
- 25 Meng He, J. Ian Munro, and Srinivasa Satti Rao. Succinct ordinal trees based on tree covering. *ACM Transactions on Algorithms*, 8(4):1–32, 2012. doi:10.1145/2344422.2344432.
- 26 Hsien-Kuei Hwang and Ralph Neininger. Phase change of limit laws in the quicksort recurrence under varying toll functions. *SIAM Journal on Computing*, 31(6):1687–1722, January 2002. doi:10.1137/s009753970138390x.
- 27 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, A001263, 2021. URL: <https://oeis.org/A001263>.
- 28 G. Jacobson. Space-efficient static trees and graphs. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1989. doi:10.1109/sfcs.1989.63533.
- 29 Jesper Jansson, Kunihiro Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, March 2012. doi:10.1016/j.jcss.2011.09.002.

- 30 J.C. Kieffer and En-Hui Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, May 2000. doi:10.1109/18.841160.
- 31 John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *2009 IEEE International Symposium on Information Theory*. IEEE, June 2009. doi:10.1109/isit.2009.5205704.
- 32 Markus Lohrey. Grammar-based tree compression. In *International Conference on Developments in Language Theory*, pages 46–57. Springer, 2015.
- 33 J. Ian Munro and Sebastian Wild. Entropy trees and range-minimum queries in optimal average-case space, 2019. arXiv:1903.02533.
- 34 Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.
- 35 Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2):29:1–29:36, February 2021.
- 36 Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Computing Surveys*, 54(2):26:1–26:38, February 2021. doi:10.1145/3432999.
- 37 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2, April 2007. doi:10.1145/1216370.1216372.
- 38 Jürg Nievergelt and Edward M. Reingold. Binary search trees of bounded balance. *SIAM J. Comput.*, 2(1):33–43, 1973. doi:10.1137/0202005.
- 39 Nicola Prezza. Optimal Rank and Select Queries on Dictionary-Compressed Text. In Nadia Pisanti and Solon P. Pissis, editors, *Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of *LIPIcs*, pages 4:1–4:12, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CPM.2019.4.
- 40 Louisa Seelbach Benkner and Markus Lohrey. Average Case Analysis of Leaf-Centric Binary Tree Sources. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *LIPIcs*, pages 16:1–16:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl. doi:10.4230/LIPIcs.MFCS.2018.16.
- 41 Dekel Tsur. Representation of ordered trees with a given degree distribution, 2018. arXiv:1807.00371.
- 42 Sebastian Wild. *Dual-Pivot Quicksort and Beyond: Analysis of Multiway Partitioning and Its Practical Potential*. Dissertation (Ph. D. thesis), University of Kaiserslautern, 2016. URL: <https://www.wild-inter.net/publications/wild-2016>.
- 43 Jie Zhang, En-Hui Yang, and John C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Transactions on Information Theory*, 60(3):1373–1386, March 2014. doi:10.1109/tit.2013.2295392.

A Range-Minimum Queries With Runs

In this appendix, we give the proofs of the results from Section 5.3.

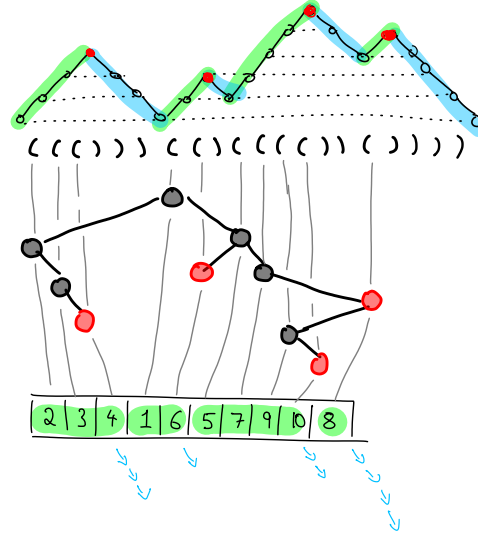
A.1 Lower Bound

In this section, we proof Theorem 6.

We refer to a run of length one as a singleton run. The types of nodes in the Cartesian tree (whether or not their left and right children exist) directly reflect their role in runs: A binary node is a run head of a non-singleton run, a leaf is the last node of non-singleton run, a right-unary node (i.e., unary node with a right child) is a middle node of run and a left-unary node is a singleton run. The leftmost node, i.e., the node with smallest inorder rank, is the only exception to this rule: if the leftmost run is a singleton run, the leftmost node is a leaf; otherwise it is right-unary.

In any case, a Cartesian tree for an array with r runs that has b binary nodes and u_ℓ left-unary nodes thus satisfies $r = b + u_\ell + 1$: every binary node represents the non-singleton run that begins with it, every left-unary node represents the singleton run at that position, and the leftmost run is counted separately. (Note that we do not double count the latter because the leftmost node is by definition neither binary nor left-unary.) We therefore obtain a lower bound for the number of equivalence classes among length- n arrays with r runs under range-minimum queries by counting binary trees with a given number of nodes n and a given number of nodes of certain types.

That r is the sum of two quantities is inconvenient, hence we instead consider the following sequence of bijections (see Figure 2). First, we map Cartesian trees t of n nodes bijectively to balanced-parenthesis (BP) strings of n pairs of parentheses as follows: The empty tree corresponds to the empty string. For a nonempty tree, we recursively compute the BP strings of the (potentially empty) left resp. right subtrees of the root; let these be denoted by L and R . Then the BP string for the entire tree is obtained as $L(R)$. (This is a variation of the canonical BP representation.)



■ **Figure 2** An example illustrating the bijections: The input array is $A = (2, 3, 4, 1, 6, 5, 7, 9, 10, 8)$, the min-oriented Cartesian tree is shown above with run ends highlighted in red. The BP string for the Cartesian tree is shown above the tree, with tree nodes connected to the corresponding opening parenthesis (note that nodes appear in inorder in the BP string). The mountain-valley (excess, Dyck path) representation of the BP string is on top; run ends correspond to peaks there.

It is easy to check that the resulting sequence is indeed the push/pop sequence of a max-stack [9, 19] where ‘(’ means push and ‘)’ means pop. We map this sequence to a lattice path by replacing ‘(’ by step vector $(1, +1)$ and ‘)’ by $(1, -1)$; the resulting lattice path is a mountain-valley diagram (Dyck paths).

The important property of the above bijections is that they *preserve runs*: A *run end* is an index where the next number is smaller (or nonexistent). In the Cartesian tree, these are the leaves *and* left-unary nodes, in the BP string, these are the occurrences of ‘(’ and in the mountain-valley representations, these are the *peaks*. The latter is known to be counted by the *Narayana numbers*: There are

$$N_{n,r} = \frac{1}{n} \binom{n}{r} \binom{n}{r-1} = \frac{r}{n(n+1-r)} \binom{n}{r}^2 \quad (2)$$

mountain-valley diagrams of length $2n$ with exactly r peaks [27]. This concludes the proof of Theorem 6; the asymptotic approximation for $\lg N_{n,r}$ immediately follows from the above closed form.

A.2 Hypersuccinct RMQ with Runs

In this section, we prove Corollary 7. To this end, we show that using a hypersuccinct tree to represent the Cartesian tree of an array $A[1..n]$ with r increasing runs has a space usage that is bounded by $\lg N_{n,r} + o(n)$ bits. By Theorem 6, this space usage is optimal up to the $o(n)$ term.

As noted in Section A.1, the correspondence between runs and node types in the Cartesian tree can be made more specific by also specifying the number $s \in [r]$ of singleton runs: Singleton runs correspond to the left-unary nodes in the (min-oriented) Cartesian tree t , except possibly for a leftmost singleton run (which corresponds to a leaf). In either case, we will have $u_\ell = s \pm 1$ left-unary nodes and $\ell = r - s \pm 1$ leaves. That implies a number of binary nodes of $b = \ell - 1 \pm 1$; the remaining $u_r = n - b - \ell - u_\ell = n - 2r + s \pm 1$ nodes are right-unary nodes.

By Corollary 2, the hypersuccinct representation of the Cartesian tree t for $A[1..n]$ supports LCA-queries on t in $O(1)$ time and uses

$$|H(t)| + o(n) \leq H_0^{\text{type}}(t) + o(n)$$

bits of space. We show that $H_0^{\text{type}}(t) \leq \lg N_{n,r} + o(n)$. Let

$$p = \left(\frac{b}{n}, \frac{u_\ell}{n}, \frac{u_r}{n}, \frac{\ell}{n} \right)$$

denote the empirical distribution of node types in t , and let $H(p)$ denote the entropy of this distribution. (For probability distribution $d = (d_1, d_2, \dots, d_k)$, its entropy is defined by

$$H(d) = \sum_{i=1}^k d_i \lg \left(\frac{1}{d_i} \right),$$

as usual.)

By definition of the type-entropy H_0^{type} , we find $H_0^{\text{type}}(t) = nH(p)$. By our previous observations, p differs from

$$p' = \left(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n} \right)$$

only by $\|p - p'\|_\infty \leq \frac{2}{n}$. Using [42, Prop. 2.42], we thus find $H(p) \leq H(p') + O(n^{-0.9})$ (this follows from Hölder-continuity of $x \mapsto x \lg x$). It thus remains to show that $nH(p') \leq \lg N_{n,r} + o(n)$. By the grouping property of H , we have

$$nH(p') = n \left(H \left(\frac{r}{n}, \frac{n-r}{n} \right) + \frac{r}{n} H \left(\frac{s}{r}, \frac{r-s}{r} \right) + \frac{n-r}{n} H \left(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r} \right) \right).$$

In order to estimate the right-hand side, observe that it follows from [22, Eq. (5.22)] that $\sum_{s=0}^r \binom{r}{s} \binom{n-r}{r-s} = \binom{n}{r}$. Since all summands are positive, we have $\binom{r}{s} \binom{n-r}{r-s} \leq \binom{n}{r}$ and hence

$$\lg \binom{r}{s} + \lg \binom{n-r}{r-s} \leq \lg \binom{n}{r}, \quad \text{for all } s \in [r]. \quad (3)$$

70:18 Hypersuccinct Trees

For a number $q \in [0, 1]$, we set $h(q) = q \lg(1/q) + (1 - q) \lg(1/(1 - q))$. Using the standard inequality

$$\frac{2^{nh(q)}}{n+1} \leq \binom{n}{qn} \leq 2^{nh(q)}, \quad nq \in [0..n], \quad (4)$$

we find

$$\begin{aligned} rh\left(\frac{s}{r}\right) + (n-r)h\left(\frac{r-s}{n-r}\right) &\stackrel{(4)}{\leq} \lg\left(\frac{r}{s}\right) + \lg\left(\frac{n-r}{r-s}\right) + \lg(r+1) + \lg(n-r+1) \\ &\stackrel{(3)}{\leq} \lg\left(\frac{n}{r}\right) + \lg(r+1) + \lg(n-r+1) \\ &\stackrel{(4)}{\leq} nh\left(\frac{r}{n}\right) + \lg(r+1) + \lg(n-r+1). \end{aligned}$$

We thus have

$$\begin{aligned} nH(p') &= n\left(H\left(\frac{r}{n}, \frac{n-r}{n}\right) + \frac{r}{n}H\left(\frac{s}{r}, \frac{r-s}{r}\right) + \frac{n-r}{n}H\left(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r}\right)\right) \\ &= n\left(h\left(\frac{r}{n}\right) + \frac{r}{n}h\left(\frac{s}{r}\right) + \frac{n-r}{n}h\left(\frac{r-s}{n-r}\right)\right) \\ &\leq 2nh\left(\frac{r}{n}\right) + O(\log n) \\ &\stackrel{(4)}{\leq} 2\lg\left(\frac{n}{r}\right) + O(\log n) \\ &\stackrel{(2)}{\leq} \lg N_{n,r} + O(\log n). \end{aligned}$$

So in total, we have shown that

$$|\mathbf{H}(t)| \leq \lg N_{n,r} + o(n),$$

which implies Corollary 7.

Determining 4-Edge-Connected Components in Linear Time

Wojciech Nadara ✉

Institute of Informatics, University of Warsaw, Poland

Mateusz Radecki ✉

University of Warsaw, Poland

Marcin Smulewicz ✉

Institute of Informatics, University of Warsaw, Poland

Marek Sokołowski ✉

Institute of Informatics, University of Warsaw, Poland

Abstract

In this work, we present the first linear time deterministic algorithm computing the 4-edge-connected components of an undirected graph. First, we show an algorithm listing all 3-edge-cuts in a given 3-edge-connected graph, and then we use the output of this algorithm in order to determine the 4-edge-connected components of the graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graphs, connectivity, cuts

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.71

Related Version Full Version: <https://arxiv.org/abs/2105.01699> [33]

Funding Wojciech Nadara: Supported by ERC Starting Grant CUTACOMBS (Grant Agreement No 714704).

Marcin Smulewicz: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).

Marek Sokołowski: Supported by ERC Starting Grant BOBR (Grant Agreement No 948057).

1 Introduction

The connectivity of graphs has always been one of the fundamental concepts of graph theory. The foremost connectivity notions in undirected graphs are the k -edge-connectedness and the k -vertex-connectedness. Namely, a graph G is k -edge-connected for $k \geq 1$ if it is connected, and it remains connected after removing any set of at most $k - 1$ edges. Similarly, G is k -vertex-connected if it contains at least $k + 1$ vertices, and it remains connected after the removal of any set of at most $k - 1$ vertices.

These notions can be generalized to the graphs that are not well-connected. Namely, if H is a maximal k -vertex-connected subgraph of G , we say that H is a k -vertex-connected component of G . The edge-connected variant is, however, defined differently: we say that a pair of vertices u, v of G is k -edge-connected if it is not possible to remove at most $k - 1$ edges from G so that u and v end up in different connected components. This relation of k -edge-connectedness happens to be an equivalence relation; this yields a definition of a k -edge-connected component of G as an equivalence class of the relation. We remark that the notions of k -vertex-connected components and k -edge-connected components coincide for $k = 1$ as both simply describe the connected components of G . However, for $k \geq 2$ these definitions diverge; in particular, for $k \geq 3$ the k -edge-connected components of a graph do not even need to be connected.



© Wojciech Nadara, Mateusz Radecki, Marcin Smulewicz, and Marek Sokołowski;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 71; pp. 71:1–71:15

Leibniz International Proceedings in Informatics



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

There has been a plethora of research into the algorithms deciding the k -vertex- and k -edge-connectedness of graphs, and decomposing the graphs into k -vertex- or k -edge-connected components. However, while classical, elementary, and efficient algorithms exist for $k = 1$ and $k = 2$, these problems become increasingly more difficult for the larger values of k . In fact, even for $k = 4$, there were no known linear time algorithms to any of the considered problems. The following description presents the previous work in this area for $k \in \{1, 2, 3, 4\}$, and exhibits the related work for the larger values of k :

$k = 1$. Here, the notions of k -vertex-connectedness and k -edge-connectedness reduce to that of connectivity and the connected components of a graph. In the static setting, determining the connected components in linear time is trivial. As a consequence, more focus is being laid on dynamic algorithms maintaining the connected components of graphs. In the incremental setting, where the edges can only be added to the dynamic graph, the optimal solution is provided by disjoint-set data structures [38], which solve the problem in the amortized $\mathcal{O}(\alpha(n))$ time per query, where $\alpha(n)$ denotes the inverse of the Ackermann's fast-growing function. The fully dynamic data structures are also considered [44, 7, 9, 18, 19, 20, 40, 27, 22, 25].

$k = 2$. One step further are the notions of 2-vertex-connectivity (biconnectivity) and 2-edge-connectivity. In the static setting, partitioning of a graph into 2-vertex-connected or 2-edge-connected components are classical problems, both solved in linear time by exploiting the properties of the low function [23]. The incremental versions of both problems are again solved optimally in the amortized $\mathcal{O}(\alpha(n))$ time per query [43]. Significant research has been done in the dynamic setting as well [36, 20, 7, 22, 11, 17, 31].

$k = 3$. As a next step, we consider 3-vertex-connectivity (triconnectivity) and 3-edge-connectivity. An optimal, linear time algorithm detecting the 3-vertex-connected components was first given by Hopcroft and Tarjan [24]. The first linear time algorithm for 3-edge-connectivity was discovered much later by Galil and Italiano [12], where they present a linear time reduction from the k -edge-connectivity problem to k -vertex-connectivity for $k \geq 3$, showing that in the static setting, the former problem is the easier of the two. This was later followed by a series of works simplifying the solution for 3-edge-connectivity [41, 35, 34, 42]. The incremental setting [32, 21] and the dynamic setting [11, 7] were also considered.

We also mention that in the case of 3-vertex-connectivity, there exists a structure called SPQR-tree which succinctly captures the structure of 2-vertex-cuts in graphs [4, 21]. Its edge-connectivity analogue also exists, but we defer its introduction to the general setting.

$k = 4$. We move on to the problems of 4-vertex-connectivity and 4-edge-connectivity. A notable result by Kanevsky et al. [26] supports maintaining 4-vertex-connected components in incremental graphs, with an optimal $\mathcal{O}(\alpha(n))$ amortized time per query. Their result also yields the solution for static graphs in $\mathcal{O}(m + n\alpha(n))$ time complexity. By applying the result of Galil and Italiano [12], we derive a static algorithm determining the 4-edge-connected components in the same time complexity. This algorithm is optimal for $m = \Omega(n\alpha(n))$.

Another result by Dinitz and Westbrook [6] supports maintaining the 4-edge-connected components in the incremental setting. Their algorithm processes any sequence of queries in $\mathcal{O}(q + m + n \log n)$ time where q is the number of queries, and m is the total number of inserted edges to the graph.

However, it is striking that the fastest solutions for 4-edge-connectivity and 4-vertex-connectivity for static graphs were derived from the on-line algorithms working in the incremental setting. In particular, no linear time algorithms for $k = 4$ were known before.

$k \geq 5$. As a side note, we also present the current knowledge on the general problems of k -vertex-connectivity and k -edge-connectivity. A series of results [29, 30, 16, 2] show that it is possible to compute the minimum edge cut of a graph (i.e., determine the edge-connectivity of a graph) in near-linear time. The previously mentioned work by Dinitz and Westbrook [6] maintains the k -edge-connected components of an incremental graph which is assumed to already have been initialized with a $(k-1)$ -edge-connected graph. The data structure answers any sequence of on-line queries in $\mathcal{O}(q + m + k^2 n \log(n/k))$ time, where q is the number of queries, and m is the number of edges in the initial graph.

Gomory and Hu [13] proved that for any weighted, undirected graph G there exists a weighted, undirected tree T on the same vertex set such that for any two vertices $s, t \in V(G)$, the value of the minimum s - t edge cut in T is equal to the value of the minimum s - t edge cut in G . Moreover, such a tree can be constructed using $n-1$ invocations of the maximum flow algorithm. In an interesting result by Hariharan et al. [15], the decomposition of any graph into k -edge-connected components is constructed in $\mathcal{O}((m + nk^3) \cdot \text{polylog}(n))$ time, producing a partial Gomory-Hu tree as its result.

Dinitz et al. [5] showed that the set of all minimum edge cuts can be succinctly represented with a *cactus graph*. When the minimum edge cut is odd, this cactus simplifies to a tree (see [8, Corollary 8]). These results imply that if the size of the minimum cut is odd, then the number of minimum cuts is $\mathcal{O}(n)$ and if it is even, then the number of minimum cuts is $\mathcal{O}(n^2)$. The structure of k -vertex-cuts was also investigated [37].

Our results. In this work, we present a linear time, deterministic algorithm partitioning static, undirected graphs into 4-edge-connected components. Even though the area of the dynamic versions of the algorithms for k -edge-connectivity is still thriving, the progress in static variants appears to have plateaued. In particular, both subquadratic algorithms determining the 4-edge-connected components [26, 6] originate from their dynamic incremental equivalents and are almost thirty years old, yet they did not achieve the optimal linear running time. Hence, our work constitutes the first progress in the static setting of 4-edge-connectivity in a long time. As a side result, our algorithm also produces the tree representation of 3-edge-cuts as explained in [8].

Organization of the work. In Section 3 [33], omitted in this abridged version of the work, we show how to reduce the problem of determining 4-edge-connected components to the problem of determining 4-edge-connected components in 3-edge-connected graphs. In Section 4, we show a linear time, randomized Monte Carlo algorithm for listing all 3-edge-cuts in 3-edge-connected graphs. In Section 5, we show how to remove the dependency on the randomness in the algorithm from the previous section, producing a linear time, deterministic algorithm listing all 3-edge-cuts in 3-edge-connected graphs. Then, in Section 6, we construct a tree of 3-edge-cuts in a 3-connected graph, given the list of all its 3-edge-cuts. This tree is then used to determine the 4-edge-connected components of the graph. Finally, in Section 7, we present open problems related to this work.

Section 3 and the proofs of the results marked ★ are deferred to the full version of the work [33] due to the space constraints.

2 Preliminaries

Graphs. In this work, we consider undirected, connected graphs which may contain self-loops and multiple edges connecting pairs of vertices (i.e., multigraphs). The number of vertices of a graph and the number of its edges are usually denoted n and m , respectively.

We use the notions of k -edge-connectedness and k -edge-connected components defined in Section 1. Moreover, we say that a set of k edges of a graph forms a k -edge-cut (or a k -cut for simplicity) if the removal of these edges from the graph disconnects it.

DFS trees. Consider a run of the *depth-first search* algorithm [39] on a connected graph G . A *depth-first search tree* (or a *DFS tree*) is a spanning tree \mathcal{T} of G , rooted at the source of the search r , containing all the edges traversed by the algorithm. After the search is performed, each vertex v is assigned two values: its *preorder* $\text{pre}(v)$ (also called *discovery time* or *arrival time*) and *postorder* $\text{post}(v)$ (also *finishing time* or *departure time*). Their definitions are standard [3]; it can be assumed that the values range from 1 to $2n$ and are pairwise different.

The edges of \mathcal{T} are called *tree edges*, and the remaining edges are called *back edges* or *non-tree edges*. In this setup, every back edge e connects two vertices remaining in ancestor-descendant relationship in \mathcal{T} ; moreover, the graph $\mathcal{T} + e$ contains exactly one cycle, named the *fundamental cycle* of e . For a vertex v of G , we define \mathcal{T}_v to be the subtree of \mathcal{T} rooted at v ; similarly, for a tree edge e whose deeper endpoint is v , we set $\mathcal{T}_e = \mathcal{T}_v$.

When a DFS tree \mathcal{T} of G is fixed, it is common to introduce directions to the edges of the graph: all tree edges of \mathcal{T} are directed away from the root of \mathcal{T} , and all back edges are pointed towards the root of \mathcal{T} . Then, uv is a directed edge (either a tree or a back edge) whose *origin* (or *tail*) is u , and whose *destination* (or *head*) is v .

For our convenience, we introduce the following definition: a back edge $e = pq$ *leaps over* a vertex v if $p \in \mathcal{T}_v$, but $q \notin \mathcal{T}_v$; we analogously define *leaping over* a tree edge f .

Moreover, we define a partial order $\leq_{\mathcal{T}}$ on the vertices of G and the tree edges of \mathcal{T} as follows: $x \leq_{\mathcal{T}} y$ if the simple path in \mathcal{T} connecting the root of \mathcal{T} with y also contains x . Then, $\leq_{\mathcal{T}}$ has one minimal element – the root of \mathcal{T} – and each maximal element is a leaf of \mathcal{T} . When the tree \mathcal{T} is clear from the context, we may write \leq instead of $\leq_{\mathcal{T}}$. We may also use $x < y$ for $x \leq y \wedge x \neq y$. Using the precomputed preorder and postorder values in \mathcal{T} , we can verify if $x \leq_{\mathcal{T}} y$ holds for given $x, y \in V(G) \cup E(\mathcal{T})$ in constant time.

We use the classical low function defined by Hopcroft and Tarjan [23]. However, for our purposes it is more convenient to define it as a function $\text{low} : E(\mathcal{T}) \rightarrow (E \setminus E(\mathcal{T})) \cup \{\perp\}$ such that for a tree edge e , $\text{low}(e)$ is the back edge uv leaping over e minimizing the preorder of its head v , breaking ties arbitrarily; or \perp , if no such edge exists. This function can be computed for all tree edges in time linear with respect to the size of the graph.

Xors. For sets A and B , by $A \oplus B$ we denote their symmetric difference, which is $(A \cup B) \setminus (A \cap B)$, and we call it a *xor* of A and B . Moreover, for a pair of non-negative integers a and b , by $a \oplus b$ we denote their xor, that is, an integer whose binary representation is a bitwise symmetric difference of the binary representations of a and b . The definitions can be easily generalized to the symmetric differences of multiple sets or integers.

4 Simple randomized algorithm

In this section, we will describe a randomized linear time algorithm listing 3-edge-cuts in 3-edge-connected graphs. In particular, the existence of this algorithm will imply that the number of 3-edge-cuts in any 3-edge-connected graph is at most linear. Since the algorithm is significantly simpler than its deterministic variant, and it already contains most of the core ideas of this work, we believe it serves as a good intermediate step in the explanation.

The overview of the algorithm is as follows. First, we will construct a randomized oracle (Lemma 5) that given two edges e, f of the 3-edge-connected graph G , verifies in constant time whether there exists another edge g such that $\{e, f, g\}$ is a 3-edge-cut of G ; and if such

an edge exists, it is returned by the oracle. It will be apparent from the description of the oracle that such an edge – if it exists – is unique. Thanks to Lemma 5, the description of our randomized algorithm will be simplified significantly: now, we only need to identify two out of three edges of each 3-edge-cut of G . Then, the remaining edge of each cut can be easily recovered using a single oracle call.

Next, our algorithm will consider an arbitrary DFS tree \mathcal{T} of G , categorize all 3-edge-cuts by the number $t \in \{0, 1, 2, 3\}$ of tree edges of \mathcal{T} in the cut, and find all 3-edge-cuts separately for each value of t . The case $t = 0$ is trivial: \mathcal{T} is a spanning tree of G , hence each edge cut must contain at least one edge of \mathcal{T} . The cases $t = 1$ and $t = 2$ will proceed by considering all tree edges of \mathcal{T} separately. For each such tree edge e , we will find all 3-edge cuts containing both e and some non-tree edge leaping over e . By performing a case study, we will show that for each e , there is only a constant number of 3-edge-cuts of this form for $t \in \{1, 2\}$; and moreover, each of them can be located by us in constant time after a linear time preprocessing of \mathcal{T} . Finally, the case $t = 3$ will be solved by a recursive call to the algorithm on a properly contracted graph.

We begin the detailed explanation with the description of some auxiliary data structures.

► **Theorem 1** ([10]). *There exists a data structure for the disjoint set union problem which, when initialized with an undirected tree \mathcal{T} (the “union tree”) on n vertices, creates n singleton sets. After the initialization, the data structure accepts the following queries in any order:*

- *find(x): returns the index of the set containing x ,*
- *union(x, y): if x and y are in different sets, then an arbitrary one of them is replaced with their union and the other one with the empty set. This query can only be issued if xy is an edge of \mathcal{T} .*

The data structure executes any sequence of q queries in total $\mathcal{O}(n + q)$ time.

► **Lemma 2** (★). *It is possible to enrich the data structure from Theorem 1, so that after rooting it at an arbitrary vertex, we are able to answer the following query in constant time:*

- *lowest(x): returns the smallest vertex of the set containing x with respect to $\leq_{\mathcal{T}}$.*

Note that each set induces a connected subgraph of \mathcal{T} , so its smallest vertex is well-defined.

► **Theorem 3** (★). *There exists a deterministic algorithm that takes as input:*

- *an undirected, unrooted tree \mathcal{T} with n vertices,*
- *p weighted paths P_1, P_2, \dots, P_p in the tree, where the path P_i has weight $w_i \in \{0, 1, \dots, C\}$,*
- *and a positive integer k ,*

and for each edge e of the tree returns the indices of k paths with the lowest weight containing e , breaking ties arbitrarily; if e is a part of fewer than k paths, all such paths are returned. The time complexity of the algorithm is $\mathcal{O}(nk + p + C)$.

We proceed to the description of our randomized algorithm. Let us choose an arbitrary vertex r of the graph, and perform a depth-first search from r . Let \mathcal{T} be the resulting DFS tree. We are now going to define a *hashing function* $H : E \rightarrow \mathcal{P}(E)$, where $\mathcal{P}(E)$ denotes the powerset of E ; the value $H(e)$ will be called a *hash* of e . If $e \notin \mathcal{T}$, then we define $H(e) = \{e\}$. Otherwise, we take $H(e)$ as the set of non-tree edges leaping over e . Let us note that the set C_e of edges f such that $e \in H(f)$ forms a cycle – the fundamental cycle of e .

► **Lemma 4** (★). *For a connected graph $G = (V, E)$ and a subset A of its edges, the graph $G' := G - A$ is disconnected if and only if there is a nonempty subset $B \subseteq A$ such that *xor* of the hashes of the edges in B is an empty set.*

Since our graph has no 1-edge-cuts or 2-edge-cuts, no created hashes are empty, and no two edges have equal hashes. Hence, removing a set of three edges disconnects a 3-edge-connected graph if and only if xor of the hashes of all of them is the empty set. Moreover, after removing some 3-edge-cut, the graph disconnects into exactly two components, and no removed edge connects vertices within one component.

As storing hashes as sets of edges would be inefficient, we define *compressed hashes*, resembling the notion of sketches introduced by Ahn et al. [1] and Kapron et al. [28]. We express compressed hashes as b -bit numbers where $b = \lceil 3 \log_2(m) \rceil$, i.e., as a function $CH : E \rightarrow \{0, \dots, 2^b - 1\}$. For each non-tree edge e , we draw $CH(e)$ randomly and uniformly from the set of b -bit numbers. For a tree edge e , we define its compressed hash $CH(e)$ as the xor of the compressed hashes of the edges in its hash, i.e., $CH(e) = \bigoplus_{f \in H(e)} CH(f)$. Note that since $b = \mathcal{O}(\log m)$, we can perform arithmetic operations on compressed hashes in constant time. However, this comes at a cost of allowing the collisions of compressed hashes.

For the ease of exposition, in the description of this algorithm we will use hash tables; because of that, the algorithm described in this section will have expected linear instead of worst-case linear running time. Formally, we create a hash table $M : \{0, \dots, 2^b - 1\} \rightarrow E \cup \{\perp\}$, which for any b -bit number x returns an edge whose compressed hash is equal to x ; or \perp if no such edge exists. In the unlikely event that there are multiple edges whose compressed hashes are equal to x , M returns any of them.

We will now categorize 3-edge-cuts based on the number of tree edges they contain, and show how to handle each case. We remark that each 3-edge-cut must intersect \mathcal{T} as it is a spanning tree of G . Therefore, a 3-edge-cut may intersect \mathcal{T} in one edge (Subsection 4.1), two edges (Subsection 4.2), or three edges (Subsection 4.3).

Throughout the case analysis, we will heavily rely on the following fact:

► **Lemma 5.** *Given two edges e and f of some 3-edge-cut in a 3-edge-connected graph, the remaining edge is uniquely identified by its hash: $H(e) \oplus H(f)$.*

Proof. If $\{e, f, g\}$ is a 3-edge-cut, then $H(e) \oplus H(f) \oplus H(g) = \emptyset$, so $H(g) = H(e) \oplus H(f)$. ◀

4.1 One tree edge

Let e be an edge of \mathcal{T} that is the only tree edge of some 3-edge-cut. The two resulting connected components after the removal of such a cut are \mathcal{T}_e and $\mathcal{T} \setminus \mathcal{T}_e$. As e is not a bridge, $\text{low}(e)$ is well-defined and connects \mathcal{T}_e with $\mathcal{T} \setminus \mathcal{T}_e$, so it belongs to the cut as well. By Lemma 5, this uniquely determines the third edge of this cut. Thus, in order to detect all such cuts, we iterate over all tree edges e and for each of them, we look up in M if there exists an edge with compressed hash equal to $CH(e) \oplus CH(\text{low}(e))$. If it exists and if it turns out to be a non-tree edge g , we output the triple $\{e, \text{low}(e), g\}$ as a 3-edge-cut.

4.2 Two tree edges

Let e and f be the edges of \mathcal{T} that form a 3-edge-cut c together with some back edge g . Without loss of generality, we can assume that $e <_{\mathcal{T}} f$, thanks to the following fact:

► **Lemma 6 (★).** *The edges e and f are comparable with respect to $\leq_{\mathcal{T}}$.*

The two connected components of $G \setminus c$ are $L := \mathcal{T}_e \setminus \mathcal{T}_f$ and $R := \mathcal{T} \setminus L$, and the remaining back edge g connects L and R . We distinguish two cases, differing in the location of g (see Figure 2 from the full version of this work): *two tree edges, lower case*, where g connects L with \mathcal{T}_f , and *two tree edges, upper case*, where g connects L with $\mathcal{T} \setminus \mathcal{T}_e$.

Two tree edges, lower case. Let A be the set of back edges between \mathcal{T}_f and $\mathcal{T} \setminus \mathcal{T}_f$. It consists of the edge g , connecting L with \mathcal{T}_f , and of several edges connecting \mathcal{T}_f with $\mathcal{T} \setminus \mathcal{T}_e$. Let B be the set of heads of edges from A (we remind that back edges are directed towards the root r). All elements of B lie on the path from f to the root r , and the head of g is the deepest element of B .

Using this observation, we will use the algorithm from Theorem 3. We initialize an instance of it with the tree \mathcal{T} , $k = 1$, $C = 2n$. Then, for each back edge $e = xy$, we create one input path P_e from x to y of weight $w_e := 2n - \text{pre}(y)$. The algorithm determines, for each tree edge f , the back edge leaping over f with the largest preorder of its head; call this edge $\text{MaxUp}(f)$. This back edge is the only candidate for g , given f .

Hence, we can find all such cuts by firstly initializing the data structure, and then iterating over all tree edges f . For each f , we take $g = \text{MaxUp}(f)$. Knowing f and g , we can look up in M whether there exists an edge whose compressed hash is $CH(f) \oplus CH(g)$ (Lemma 5). If it exists and if it is a tree edge, then we call it e and output the triple $\{e, f, g\}$ as a 3-edge-cut.

Two tree edges, upper case. Let A be the set of non-tree edges between \mathcal{T}_e and $\mathcal{T} \setminus \mathcal{T}_e$. It consists of the back edge g , connecting L with $\mathcal{T} \setminus \mathcal{T}_e$, and of several edges connecting \mathcal{T}_f with $\mathcal{T} \setminus \mathcal{T}_e$. Let B be the set of the tails of the edges from A . Let $v \in L$ be the tail of g . As $v \notin \mathcal{T}_f$, then we either have $\text{pre}(v) < \min_{u \in \mathcal{T}_f} \text{pre}(u)$, or $\text{pre}(v) > \max_{u \in \mathcal{T}_f} \text{pre}(u)$. In the first case, v is the vertex with the smallest preorder which is a tail of some edge leaping over e ; while in the second case v is the analogous vertex with the largest preorder.

Based on this observation, we will again use the algorithm from Theorem 3 again. We initialize one instance of it with the tree \mathcal{T} , $k = 1$, and $C = 2n$. Then for each back edge $e = xy$, we create one input path P_e from x to y of weight $w_e := \text{pre}(x)$. We also initialize another instance of this algorithm in the same way, only that we set $w_e := 2n - \text{pre}(x)$ instead. The first instance determines, for each tree edge e , the back edge $\text{MinDn}(e)$ leaping over e with the smallest preorder of its tail; while the second instance determines the analogous edge $\text{MaxDn}(e)$, with the largest preorder of its tail. By our considerations above, if any desired $\{e, f, g\}$ cut exists, then $g \in \{\text{MinDn}(e), \text{MaxDn}(e)\}$.

Hence, we can find all such cuts by firstly initializing both instances of the data structure. Then, we iterate over all tree edges e , and for each e we check both candidates for g . If our hash table contains an edge whose compressed hash is $CH(e) \oplus CH(g)$ and it is a tree edge, then we call it f and output the triple $\{e, f, g\}$ as a 3-edge-cut.

4.3 Three tree edges

Solving this case in a way similar to the previous cases seems intractable. We consider this subsection, together with the time analysis following it, as one of the key ideas of this work.

In this case, we assume that no non-tree edges belong to the cut. Therefore, we can contract all of them simultaneously and recursively list all 3-edge-cuts in the resulting graph. Since 3-edge-cuts in the contracted graph exactly correspond to the 3-edge-cuts consisting solely of tree edges in the original graph, this reduction is sound. The contraction is performed in $\mathcal{O}(n + m)$ time by identifying the vertices within the connected components of $G - E(\mathcal{T})$. Let G' be the graph after these contractions. Since G is 3-edge-connected, G' is 3-edge connected as well, so the assumption about the input graph being 3-edge-connected is preserved. We do not modify the value of b in the subsequent recursive calls, even though the value of m decreases.

4.4 Time and correctness analysis

We will now compute the expected time complexity of our algorithm. The subroutines from Subsections 4.1 and 4.2 clearly take expected linear time. The only nontrivial part of the analysis is the recursion in Subsection 4.3. Let $T(m)$ denote the maximum expected time our algorithm needs to solve any graph with at most m edges. Since our graph is 3-edge-connected, the degree of each vertex in G is at least 3, so $|E(G)| \geq \frac{3}{2}|V(G)| \geq \frac{3}{2}|E(G')| \Rightarrow |E(G')| \leq \frac{2}{3}|E(G)|$. Therefore, we have that $T(m) \leq \mathcal{O}(m) + T(\frac{2}{3}m)$. The solution to this recurrence is $T(m) = \mathcal{O}(m)$, hence the whole algorithm runs in expected linear time.

We proceed to proving that our algorithm works correctly with sufficiently high probability. The only reason it can output wrong result comes from the compression of hashes – if we used their uncompressed version instead, the algorithm would clearly be correct.

The maximum number of queries to our hash table is linear in terms of n , which follows from a similar argument to the one presented in Subsection 4.4. Hence, there exists some absolute constant c such that the number of queries is bounded by cn . For each query with value q , and for each edge whose hash is not equal to the hash whose compressed version we ask about, there is 2^{-b} probability that compressed version of this hash is equal to q . Hence, the probability that we ever get a false positive is bounded from above by $cnm2^{-b}$. Since $b = \lceil 3\log_2(m) \rceil$ and $n \leq m$, we infer that $cnm2^{-b} \leq \frac{c}{m}$. Therefore, our algorithm works correctly with probability at least $1 - \frac{c}{m}$.

5 Deterministic algorithm

Having established a linear time randomized algorithm producing 3-edge-cuts in 3-edge-connected graphs, we will now determinize it by designing deterministic implementations of the subroutines for each of the cases considered in the randomized algorithm. Three cases need to be derandomized: “one tree edge” (Subsection 4.1), “two tree edges, lower case”, and “two tree edges, upper case” (Subsection 4.2). In the following description, we cannot use compressed hashes anymore: the compression is a random process which inevitably results in false positives. Instead, we will exploit additional properties of 3-edge-cuts in order to produce an efficient deterministic implementation of the algorithm.

Recall that in the description of the randomized implementation of the algorithm, we defined the values $\text{low}(e)$, $\text{MaxUp}(e)$, $\text{MinDn}(e)$, and $\text{MaxDn}(e)$ for any tree edge e . For the deterministic variant of the algorithm, we generalize these notions: we define $\text{low1}(e)$, $\text{low2}(e)$, and $\text{low3}(e)$ as the three back edges leaping over e with the minimal preorders of their targets; in particular, we set $\text{low1}(e) := \text{low}(e)$. We analogously define $\text{MaxUp1}(e)$, $\text{MaxUp2}(e)$, $\text{MinDn1}(e)$, $\text{MinDn2}(e)$, $\text{MaxDn1}(e)$, and $\text{MaxDn2}(e)$. We remark that $\text{low3}(e)$ might not exist if there are fewer than three edges leaping over e ; in this case, we put $\text{low3}(e) := \perp$. However, all the other values must exist – otherwise, at most one edge would leap over e , which would mean that this edge, together with e , would form a 2-edge-cut of G .

It is straightforward to compute all the values defined above in linear time; for instance, the generalizations of MaxUp , MinDn , and MaxDn can be determined by passing $k = 2$ to the algorithm in Theorem 3 instead of $k = 1$. Hence, in the following description, we will assume that all the values above have already been computed.

5.1 One tree edge

Recall that in this case, we are to find all 3-edge-cuts intersecting a fixed depth-first search tree \mathcal{T} in a single edge. This is fairly straightforward: if some 3-edge-cut C contains exactly one tree edge e of \mathcal{T} , then the border of the cut is exactly \mathcal{T}_e ; hence, this cut must include

all back edges leaping over e . Therefore, e belongs to a 3-edge-cut of this kind if and only if $\text{low3}(e) = \perp$, i.e. if there only exist two back edges leaping over e ; in this case, e and these two edges form a 3-edge-cut. Since this check can be easily performed in $\mathcal{O}(n)$ time for all tree edges in \mathcal{T} , the whole subroutine runs in $\mathcal{O}(n + m)$ time complexity.

5.2 Two tree edges, lower case

Recall that this case requires us to find all 3-edge-cuts intersecting a fixed depth-first search tree \mathcal{T} in two edges, say e and f , such that $e < f$ and the remaining back edge g connects \mathcal{T}_f with $\mathcal{T}_e \setminus \mathcal{T}_f$. Hence, g is the only back edge connecting \mathcal{T}_f with $\mathcal{T}_e \setminus \mathcal{T}_f$, no back edges connect $\mathcal{T}_e \setminus \mathcal{T}_f$ with $\mathcal{T} \setminus \mathcal{T}_e$, but there may be multiple back edges connecting \mathcal{T}_f with $\mathcal{T} \setminus \mathcal{T}_e$.

We shall exploit the fact that in a valid 3-edge-cut of this kind, all back edges leaping over e must originate from \mathcal{T}_f . This observation severely limits the set of possible tree edges f . This is formalized by the notion of the *deepest down cut* of e :

► **Definition 7** (deepest down cut). *For a tree edge e in \mathcal{T} , we define the deepest down cut of e , denoted $\text{DeepestDnCut}(e)$, as the deepest tree edge $f \geq e$ for which all back edges leaping over e originate from \mathcal{T}_f .*

► **Lemma 8** (★). *For every tree edge e , $\text{DeepestDnCut}(e)$ is defined correctly and uniquely. Moreover, every 3-edge-cut $\{e, f, g\}$ of the considered kind satisfies $e < f \leq \text{DeepestDnCut}(e)$.*

► **Lemma 9** (★). *For every tree edge e , $\text{DeepestDnCut}(e) = uv$ is the tree edge whose head v is the lowest common ancestor of two vertices: the tails of $\text{MinDn1}(e)$ and $\text{MaxDn1}(e)$.*

Thanks to Lemma 9, we can compute $\text{DeepestDnCut}(e)$ in constant time for each tree edge e as the lowest common ancestor of two vertices can be computed in constant time after linear preprocessing [14].

We now shift our focus to the lower tree edge f . As in Section 4, the only possible candidate g for a back edge of the cut leaping over f is given by $\text{MaxUp1}(f)$. The only problematic part is locating the remaining tree edge e . Previously, we utilized randomness in order to calculate the compressed hash of e given the compressed hashes of f and g . Here, we instead use the following fact:

► **Lemma 10** (★). *If $\{e, f, g\}$ is a 3-edge-cut of the considered kind for some tree edges e and $f > e$, then e is the deepest tree edge satisfying $\text{DeepestDnCut}(e) \geq f$.*

Lemmas 8 and 10 naturally lead to the following idea: given a tree edge e , the set of edges f satisfying $\text{DeepestDnCut}(e) \geq f$ is a path P_e connecting the head of e with the head of $\text{DeepestDnCut}(e)$; we assign this path a weight w_e equal to the depth of e in \mathcal{T} . We then invoke Theorem 3 with the tree \mathcal{T} , the weighted paths $\{P_e \mid e \in E(\mathcal{T})\}$, and $k = 1$. This lets us find, in $\mathcal{O}(n)$ time, for each tree edge $f \in E(\mathcal{T})$, the path P_e of the maximum weight containing f as an edge. This path naturally corresponds to the tree edge e from Lemma 10. This way, for every tree edge f , we have uniquely identified a back edge g and a tree edge e such that the only possible 3-edge-cut containing f as a deeper tree edge is $\{e, f, g\}$.

It only remains to verify that $\{e, f, g\}$ is a 3-edge-cut. Since $\text{DeepestDnCut}(e) \geq f$ guarantees that all back edges leaping over e originate from \mathcal{T}_f , we only need to check that exactly one edge (that is, g) connects \mathcal{T}_f with $\mathcal{T}_e \setminus \mathcal{T}_f$. As $g = \text{MaxUp1}(f)$, it suffices to verify that the edge $\text{MaxUp2}(f)$, which is a back edge leaping over f whose head is the deepest apart from g , also leaps over e (i.e., it does not terminate in \mathcal{T}_e).

It can be easily verified that the implementation of the subroutine is deterministic and runs in linear time with respect to the size of G .

5.3 Two tree edges, upper case

Recall that in this case, we are required to find all 3-edge-cuts intersecting \mathcal{T} in two edges e and f , such that $e < f$ and the remaining back edge g connects $\mathcal{T}_e \setminus \mathcal{T}_f$ with $\mathcal{T} \setminus \mathcal{T}_e$. Similarly to the randomized case, we use the fact that given a tree edge e , the tail of g has either the smallest preorder (i.e., $g = \text{MinDn1}(e)$) or the largest preorder (i.e., $g = \text{MaxDn1}(e)$) among all the back edges leaping over e . Without loss of generality, assume that $g = \text{MinDn1}(e)$; the latter case is analogous.

In a similar vein to previous case, observe that if $\{e, f, g\}$ is a 3-edge-cut for some $f > e$, then all back edges leaping over e other than g must originate from \mathcal{T}_f .

This leads to the following slight generalization of DeepestDnCut (Definition 7):

► **Definition 11.** For a tree edge e in \mathcal{T} , we define the value $\text{DeepestDnCutNoMin}(e)$ as the deepest tree edge $f \geq e$ for which all back edges leaping over e other than $\text{MinDn1}(e)$ originate from \mathcal{T}_f .

The process of computation of $\text{DeepestDnCut}(e)$ asserted by Lemma 9 can be easily modified to match our needs: we take $\text{DeepestDnCutNoMin}(e)$ as the tree edge whose head is the lowest common ancestor of the tails of $\text{MinDn2}(e)$ and $\text{MaxDn1}(e)$. We refer the reader to Figure 4 from the full version of the work for a helpful picture.

Now, fix the shallower tree edge e of the cut. Let $f_0 := \text{DeepestDnCutNoMin}(e)$. Then, if some tree edge $f > e$ belongs to the 3-edge-cut $\{e, f, g\}$, then $f \leq f_0$ (otherwise, there would be multiple edges connecting $\mathcal{T}_e \setminus \mathcal{T}_f$ with $\mathcal{T} \setminus \mathcal{T}_e$). The natural question is then: is $\{e, f_0, g\}$ a 3-edge-cut? The only possible problem is that there may exist back edges connecting \mathcal{T}_{f_0} with $\mathcal{T}_e \setminus \mathcal{T}_{f_0}$. Fortunately, if this is the case, then the back edge $\text{MaxUp1}(f_0)$, dependent only on f_0 , is one of these back edges.

Hence, let $h_0 := \text{MaxUp1}(f_0)$. If h_0 leaps over e , we are done, and $\{e, f_0, g\}$ is an edge cut. Otherwise, the subtree \mathcal{T}_f for the sought 3-edge-cut $\{e, f, g\}$ must contain the head of h_0 (or else h_0 would connect \mathcal{T}_f with $\mathcal{T}_e \setminus \mathcal{T}_f$). Let then $f_1, e \leq f_1 < f_0$ be the deepest tree edge containing the head of h_0 , that is, the edge whose head coincides with the head of h_0 . Then, the edge f of the 3-edge-cut $\{e, f, g\}$ must satisfy $e < f \leq f_1$. We can then repeat this procedure: given f_1 , we compute $h_1 := \text{MaxUp1}(f_1)$, and either h_1 leaps over e and we are done, or we calculate another edge $f_2 < f_1$ supplying a better bound on the depth of f .

Since the graph is finite, this process terminates in $k = \mathcal{O}(n)$ steps, producing the lowest tree edge $f_k \geq e$ such that no back edge connects \mathcal{T}_{f_k} with $\mathcal{T}_e \setminus \mathcal{T}_{f_k}$, and no back edge other than g connects $\mathcal{T}_e \setminus \mathcal{T}_{f_k}$ with $\mathcal{T} \setminus \mathcal{T}_e$. If $f_k = e$, then no 3-edge-cut $\{e, f, g\}$ exists for $f > e$. Otherwise, $\{e, f_k, g\}$ is naturally a correct 3-edge-cut. Since two edges of a 3-edge-cut uniquely identify the third edge (Lemma 5), we conclude that this is the only 3-edge-cut containing e and g . We refer the reader to Figure 5 from the full version of the paper for a helpful picture depicting the iterative process above.

In order to optimize the algorithm, we seek to optimize the iterative process described above. Indeed, for each $e \in E(\mathcal{T})$, the process is very similar: start with some edge $f > e$, and then repeatedly replace f with a higher edge, until a replacement would result in an edge closer to the root than e . We can model this process with a rooted tree \mathcal{U} defined as follows:

- the vertices of \mathcal{U} are the tree edges of \mathcal{T} , and \perp ,
- \perp is the root of \mathcal{U} ,
- for a non-root vertex e of \mathcal{U} , its parent is the tree edge with the same head as $\text{MaxUp1}(e)$.

If the head of $\text{MaxUp1}(e)$ coincides with the root of \mathcal{T} , then the parent of e in \mathcal{U} is \perp . Naturally, \mathcal{U} can be constructed in linear time with respect to the size of \mathcal{T} ; moreover, if an edge p is a parent of another edge q in \mathcal{U} , then $p <_{\mathcal{T}} q$.

Now, the iteration is equivalent to the repeated replacement of f_0 with its parent in \mathcal{U} as long as the parent is greater or equal than e with respect to $<_{\mathcal{T}}$. In other words, the final edge $f' := \text{final}(f_0, e)$ is taken as the shallowest ancestor of f_0 in \mathcal{U} for which $f' \geq_{\mathcal{T}} e$.

This leads to the final idea: we simulate a forest of rooted subtrees of \mathcal{U} using a disjoint set union data structure $F_{\mathcal{U}}$ (Theorem 1). Each subtree additionally keeps its root, which can be retrieved from $F_{\mathcal{U}}$ (Lemma 2). Initially, each subtree of \mathcal{U} contains a single vertex.

After the initialization of $F_{\mathcal{U}}$, we iterate e over the tree edges of \mathcal{T} in the decreasing order of depth in \mathcal{T} . Throughout the process, we maintain the following invariant on $F_{\mathcal{U}}$: an edge $ef \in E(\mathcal{U})$ for $e < f$ has been added to the forest if and only if e has been considered at any previous iteration as the shallower tree edge of the cut. Hence, at the beginning of the iteration for a given edge e , we add to $F_{\mathcal{U}}$ all tree edges of \mathcal{U} originating from e . At this point of time, for every tree edge f such that $f >_{\mathcal{T}} e$, the edge $\text{final}(f, e)$ is given by $F_{\mathcal{U}}.\text{lowest}(f)$. This reduces the entire iterative process described above to a single **lowest** query on $F_{\mathcal{U}}$.

We initialized $F_{\mathcal{U}}$ on a tree with n vertices, and we issued $\mathcal{O}(n)$ queries to it in total. Therefore, the whole subroutine runs in time linear with respect to the size of G .

Summing up, we replaced each randomized subroutine with its deterministic counterpart, preserving the linear guarantee on the runtime of the algorithm. We conclude that there exists a deterministic linear time algorithm listing 3-edge-cuts in 3-edge-connected graphs.

6 Reconstructing the structure of 4-edge-connected components

In this section, we show how to build a structure of 4-edge-connected components of a 3-edge-connected graph G , given the set \mathcal{C} of all 3-edge-cuts in G .

First of all, recall what such a structure looks like.

► **Theorem 12.** [8, Corollary 8] *For a 3-edge-connected graph $G = (V, E)$, there exists a tree $H = (U, F)$ and functions $\phi : \mathcal{C} \rightarrow F$ and $\psi : V \rightarrow U$, such that ϕ is a bijection from 3-edge-cuts of G to the edges of H , and ψ maps (not necessarily surjectively) vertices of G to the vertices of H so that whole 4-edge-connected components are mapped to the same vertex.*

Moreover, if a 3-edge-cut c partitions the vertices of G into two parts V_1 and V_2 , then $\phi(c)$ partitions the vertices of H into U_1 and U_2 such that $\psi^{-1}(U_1) = V_1$ and $\psi^{-1}(U_2) = V_2$.

We refer the reader to Figure 6 in the full version for an illustrative example of a decomposition postulated by Theorem 12.

The tree H is usually unrooted in the literature. However, we are going to root it. Namely, we take a depth-first search tree \mathcal{T} of G , rooted at some vertex r , and we root H at $\psi(r)$. For a vertex $u \in U$, we let H_u denote the subtree of H rooted at u .

► **Definition 13.** *For a 3-edge-cut c , we define $P(c)$ as the set of vertices from the connected component of $G \setminus c$ not containing r .*

We remark that since c is a minimal cut, $G \setminus c$ consists of two connected components, so $P(c)$ is determined uniquely.

► **Lemma 14 (★).** *Let $v, u \in U$, and let $e_1, e_2, \dots, e_k \in F$ be the sequence of edges of H on the path from v and u in H . If v is an ancestor of u , then for each pair of integers i, j such that $1 \leq i < j \leq k$, we have $|P(\phi^{-1}(e_i))| > |P(\phi^{-1}(e_j))|$.*

► **Lemma 15 (★).** *Given a 3-edge-connected graph G and the set of all its 3-edge-cuts \mathcal{C} , the sizes of $P(c)$ for all $c \in \mathcal{C}$ can be computed in linear time with respect to the size of G .*

71:12 Determining 4-Edge-Connected Components in Linear Time

To reconstruct H , we will also use the following structural lemma about cuts sharing the same edge of graph G .

► **Lemma 16 (★).** *For an edge $(u, v) = e \in E$, let $l(e)$ be the set of all 3-edge-cuts containing e . The image $\phi(l(e))$, i.e., the set of all edges of H corresponding to the edge cuts containing e , forms a path in H between $\psi(u)$ and $\psi(v)$.*

We remark an edge case in Lemma 16: if $\psi(u) = \psi(v)$ for some edge $e = (u, v)$, then the image $\phi(l(e))$ is empty. Moreover, since each 3-edge-cut $c \in \mathcal{C}$ contains at least one tree edge of \mathcal{T} , each edge $\phi(c)$ is covered by at least one path $\phi(l(e))$ for $e \in E(\mathcal{T})$. Equivalently, H is a tree, rooted at $\psi(r)$, equal to the union of all paths $\phi(l(e))$ for $e \in E(\mathcal{T})$. This representation of H is the cornerstone of our algorithm reconstructing H from G and \mathcal{C} .

► **Lemma 17.** *There exists a linear time algorithm which, given a graph G and the list \mathcal{C} of all 3-edge-cuts of G , constructs the tree H , along with the mappings ϕ and ψ .*

Sketch of the proof. For each edge $e \in \mathcal{T}$, create a list $l(e)$ of all 3-edge-cuts c containing e , sorted decreasingly by the size of $P(c)$ using radix sort. Let $e_1, \dots, e_{n-1} \in E(\mathcal{T})$ be the sequence of edges visited by a depth-first search of \mathcal{T} and let $e_i = u_i v_i$, where u_i is the vertex of \mathcal{T} closer to the root r . We create H iteratively; initially, H is a single vertex $\psi(r)$. We maintain the following invariant after k iterations of the algorithm: H is a connected tree, rooted at $\psi(r)$, equal to the union of all paths $\phi(l(e_i))$ for $i \in \{1, 2, \dots, k\}$.

It is clear that after $n - 1$ iterations, H will be the required rooted tree. Consider the k -th iteration of the algorithm, $k \in [1, n - 1]$, in which we need to add to H the path $\phi(l(e_k))$, originating from $\psi(u_k)$ and terminating at $\psi(v_k)$. It can be proved that $\psi(u_k)$, together with the vertical path connecting it with the root of H , is already in H . Now, adding the path $\phi(l(e_k))$ to H is rather straightforward: first, starting from $\psi(u_k)$, we go up the tree H along the edges of H corresponding to the edge cuts containing e_k . Then, we proceed down the tree: we iterate the list $l(e_k)$ of cuts, excluding the cuts corresponding to the edges visited in the first part of the traversal. For each such cut, we go down the tree along the edge corresponding to the cut (creating it, if necessary). Each 3-edge-cut c is considered in only a constant number of iterations of the algorithm: an edge of H corresponding to c is traversed by the path $\phi(l(e_k))$ if and only if $e_k \in c$. Therefore, the time complexity of all iterations in total is $\mathcal{O}(m + |\mathcal{C}|) = \mathcal{O}(m + n)$. ◀

This concludes the construction of a tree representing all 3-edge-cuts in G . As a result, each vertex of H , as long as it is not empty, contains a single 4-edge-connected component of G . Hence, this algorithm also computes the decomposition of a 3-edge-connected graph G into 4-edge-connected components in total linear time.

7 Open problems

As a natural open problem whose resolving would complement this result nicely, we suggest investigating if it is possible to extend our result to vertex connectivity or to the higher-order edge connectivity.

Problem 1. Given an undirected graph $G = (V, E)$, is it possible to find all 4-vertex-connected components of G in linear time?

Problem 2. Given an undirected graph $G = (V, E)$, is it possible to find all 5-edge-connected components of G in linear time?

We also remark that our algorithm assumes the word RAM model in which we can perform any arithmetic and bitwise operations on pointers and $\mathcal{O}(\log n)$ -bit words in constant time; this is required by the linear time data structure of Gabow and Tarjan (Theorem 1, [10]). The natural question is whether this assumption can be avoided.

Problem 3. Given an undirected graph $G = (V, E)$, is it possible to find all 4-edge-connected components of G in linear time in the pointer machine model?

References

- 1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing Graph Structure via Linear Measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, page 459–467, USA, 2012. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973099.40.
- 2 Nalin Bhardwaj, Antonio Molina Lovett, and Bryce Sandlund. A Simple Algorithm for Minimum Cuts in Near-Linear Time. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22–24, 2020, Tórshavn, Faroe Islands*, volume 162 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.12.
- 3 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>.
- 4 Giuseppe Di Battista and Roberto Tamassia. On-Line Graph Algorithms with SPQR-Trees. In Michael S. Paterson, editor, *Automata, Languages and Programming*, pages 598–611, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.
- 5 Yefim Dinitz, Alexander V. Karzanov, and Michael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. *Studies in Discrete Optimization*, page 290–306, 1976.
- 6 Yefim Dinitz and Jeffery R. Westbrook. Maintaining the Classes of 4-Edge-Connectivity in a Graph On-Line. *Algorithmica*, 20(3):242–276, March 1998. doi:10.1007/PL00009195.
- 7 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.
- 8 Tamás Fleiner and András Frank. A quick proof for the cactus representation of mincuts. Technical Report QP-2009-03, Egerváry Research Group, Budapest, 2009.
- 9 Greg N. Frederickson. Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. doi:10.1137/0214055.
- 10 Harold N. Gabow and Robert Endre Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.
- 11 Zvi Galil and Giuseppe F. Italiano. Fully Dynamic Algorithms for Edge Connectivity Problems. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 317–327, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103454.
- 12 Zvi Galil and Giuseppe F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, 1991. doi:10.1145/122413.122416.
- 13 Ralph E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. URL: <http://www.jstor.org/stable/2098881>.

- 14 Dov Harel and Robert Endre Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.
- 15 Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient Algorithms for Computing All Low s-t Edge Connectivities and Related Problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 127–136, USA, 2007. Society for Industrial and Applied Mathematics.
- 16 Monika Henzinger, Satish Rao, and Di Wang. *Local Flow Partitioning for Faster Edge Connectivity*, pages 1919–1938. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.125.
- 17 Monika Rauch Henzinger. Fully dynamic biconnectivity in graphs. *Algorithmica*, 13(6):503–538, June 1995. doi:10.1007/BF01189067.
- 18 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*, STOC '95, page 519–527, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/225058.225269.
- 19 Monika Rauch Henzinger and Mikkel Thorup. Sampling to Provide or to Bound: With Applications to Fully Dynamic Graph Algorithms. *Random Struct. Algorithms*, 11(4):369–379, 1997.
- 20 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.
- 21 Jacob Holm and Eva Rotenberg. Worst-Case Polylog Incremental SPQR-Trees: Embeddings, Planarity, and Triconnectivity. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 2378–2397, USA, 2020. Society for Industrial and Applied Mathematics.
- 22 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic Bridge-Finding in $\tilde{O}(\log^2 n)$ Amortized Time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 35–52, USA, 2018. Society for Industrial and Applied Mathematics.
- 23 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.
- 24 John Hopcroft and Robert Tarjan. Dividing a Graph into Triconnected Components. *SIAM Journal on Computing*, 2(3):135–158, 1973. doi:10.1137/0202012.
- 25 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully Dynamic Connectivity in $O(\log n(\log \log n)^2)$ Amortized Expected Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 510–520, USA, 2017. Society for Industrial and Applied Mathematics.
- 26 Arkady Kanevsky, Roberto Tamassia, Giuseppe Di Battista, and Jianer Chen. On-Line Maintenance of the Four-Connected Components of a Graph. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 793–801, 1991. doi:10.1109/SFCS.1991.185451.
- 27 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 '13, page 1131–1142, USA, 2013. Society for Industrial and Applied Mathematics.
- 28 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 '13, page 1131–1142, USA, 2013. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973105.81.
- 29 David R. Karger. Minimum Cuts in Near-Linear Time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

- 30 Kenichi Kawarabayashi and Mikkel Thorup. Deterministic Global Minimum Cut of a Simple Graph in Near-Linear Time. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 665–674, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746588.
- 31 Johannes A. La Poutré and Jeffery Westbrook. Dynamic 2-Connectivity with Backtracking. *SIAM J. Comput.*, 28(1):10–26, 1999. doi:10.1137/S0097539794272582.
- 32 Johannes A. La Poutré, Jan van Leeuwen, and Mark H. Overmars. Maintenance of 2- and 3-edge-connected components of graphs I. *Discrete Mathematics*, 114(1):329–359, 1993. doi:10.1016/0012-365X(93)90376-5.
- 33 Wojciech Nadara, Mateusz Radecki, Marcin Smulewicz, and Marek Sokołowski. Determining 4-edge-connected components in linear time. *CoRR*, abs/2105.01699, 2021. arXiv:2105.01699.
- 34 Hiroshi Nagamochi and Toshihide Ibaraki. A linear time algorithm for computing 3-edge-connected components in a multigraph. *Japan Journal of Industrial and Applied Mathematics*, 9(2):163, June 1992. doi:10.1007/BF03167564.
- 35 Nima Norouzi and Yung H. Tsin. A simple 3-edge connected component algorithm revisited. *Information Processing Letters*, 114(1):50–55, 2014. doi:10.1016/j.ip1.2013.09.010.
- 36 Richard Peng, Bryce Sandlund, and Daniel Dominic Sleator. Optimal Offline Dynamic 2, 3-Edge/Vertex Connectivity. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 553–565. Springer, 2019. doi:10.1007/978-3-030-24766-9_40.
- 37 Seth Pettie and Longhui Yin. The Structure of Minimum Vertex Cuts, 2021. arXiv:2102.06805.
- 38 Robert E. Tarjan and Jan van Leeuwen. Worst-Case Analysis of Set Union Algorithms. *J. ACM*, 31(2):245–281, March 1984. doi:10.1145/62.2160.
- 39 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.
- 40 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 343–350, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335345.
- 41 Yung H. Tsin. A Simple 3-Edge-Connected Component Algorithm. *Theory of Computing Systems*, 40(2):125–142, February 2007. doi:10.1007/s00224-005-1269-4.
- 42 Yung H. Tsin. Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130–146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP). doi:10.1016/j.jda.2008.04.003.
- 43 Jeffery Westbrook and Robert E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, June 1992. doi:10.1007/BF01758773.
- 44 Christian Wulff-Nilsen. Faster Deterministic Fully-Dynamic Graph Connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, page 1757–1769, USA, 2013. Society for Industrial and Applied Mathematics.

Isomorphism Testing Parameterized by Genus and Beyond

Daniel Neuen 

CISPA Helmholtz Center for Information Security,
Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We present an isomorphism test for graphs of Euler genus g running in time $2^{O(g^4 \log g)} n^{O(1)}$. Our algorithm provides the first explicit upper bound on the dependence on g for an fpt isomorphism test parameterized by the Euler genus of the input graphs. The only previous fpt algorithm runs in time $f(g)n$ for some function f (Kawarabayashi 2015). Actually, our algorithm even works when the input graphs only exclude $K_{3,h}$ as a minor. For such graphs, no fpt isomorphism test was known before.

The algorithm builds on an elegant combination of simple group-theoretic, combinatorial, and graph-theoretic approaches. In particular, we introduce (t, k) -WL-bounded graphs which provide a powerful tool to combine group-theoretic techniques with the standard Weisfeiler-Leman algorithm. This concept may be of independent interest.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graphs and surfaces

Keywords and phrases graph isomorphism, fixed-parameter tractability, Euler genus, Weisfeiler-Leman algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.72

Related Version *Full Version:* <https://arxiv.org/abs/2106.14869>

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

1 Introduction

Determining the computational complexity of the Graph Isomorphism Problem is a long-standing open question in theoretical computer science (see, e.g., [13]). The problem is easily seen to be contained in NP, but it is neither known to be in PTIME nor known to be NP-complete. In a breakthrough result, Babai [1] recently obtained a quasipolynomial-time algorithm for testing isomorphism of graphs (i.e., an algorithm running in time $n^{O((\log n)^c)}$ where n denotes the number of vertices of the input graphs, and c is a constant), achieving the first improvement over the previous best algorithm running in time $n^{O(\sqrt{n/\log n})}$ [3] in over three decades. However, it remains wide open whether GI can be solved in polynomial time.

In this work, we are concerned with the parameterized complexity of isomorphism testing. While polynomial-time isomorphism tests are known for a large variety of restricted graph classes (see, e.g., [4, 7, 9, 11, 18, 24]), for several important structural parameters such as maximum degree or the Hadwiger number¹, it is still unknown whether isomorphism testing is fixed-parameter tractable (i.e., whether there is an isomorphism algorithm running in time $f(k)n^{O(1)}$ where k denotes the graph parameter in question, n the number of vertices of the

¹ The Hadwiger number of a graph G is the maximum number h such that K_h is a minor of G .



input graphs, and f is some function). On the other hand, there has also been significant progress in recent years. In 2015, Lokshtanov et al. [17] obtained the first fpt isomorphism test parameterized by the tree-width k of the input graph running in time $2^{\mathcal{O}(k^5 \log k)} n^5$. This algorithm was later improved by Grohe et al. [8] to a running time of $2^{\mathcal{O}(k \cdot (\log k)^c)} n^3$ (for some constant c). In the same year, Kawarabayashi [14] obtained the first fpt isomorphism test parameterized by the Euler genus g of the input graph running time $f(g)n$ for some function f . While Kawarabayashi's algorithm achieves optimal dependence on the number of vertices of the input graphs, it is also extremely complicated and it provides no explicit upper bound on the function f . Indeed, the algorithm spans over multiple papers [14, 15, 16] and builds on several deep structural results for graphs of bounded genus.

In this work, we present an alternative isomorphism test for graphs of Euler genus g running in time $2^{\mathcal{O}(g^4 \log g)} n^{\mathcal{O}(1)}$. In contrast to Kawarabayashi's algorithm, our algorithm does not require any deep graph-theoretic insights, but rather builds on an elegant combination of well-established and simple group-theoretic, combinatorial, and graph-theoretic ideas. In particular, this enables us to provide the first explicit upper bound on the dependence on g for an fpt isomorphism test. Actually, the only property our algorithm exploits is that graphs of genus g exclude $K_{3,h}$ as a minor for $h \geq 4g + 3$ [25]. In other words, our main result is an fpt isomorphism test for graphs excluding $K_{3,h}$ as a minor.

► **Theorem 1.** *The Graph Isomorphism Problem for graphs excluding $K_{3,h}$ as a minor can be solved in time $2^{\mathcal{O}(h^4 \log h)} n^{\mathcal{O}(1)}$.*

For this class of graphs, the best existing algorithm runs in time $n^{\mathcal{O}((\log h)^c)}$ for some constant c [21], and no fpt isomorphism test was known prior to this work.

For the algorithm, we combine different approaches to the Graph Isomorphism Problem. On a high-level, our algorithm follows a simple decomposition strategy which decomposes the input graph G into pieces such that the interplay between the pieces is simple. The main idea is to define the pieces in such a way that, after fixing a small number of vertices, the automorphism group of G restricted to a piece $D \subseteq V(G)$ is similar to the automorphism group of a graph of maximum degree 3. This allows us to test isomorphism between the pieces using the group-theoretic graph isomorphism machinery dating back to Luks's polynomial-time isomorphism test for graphs of bounded maximum degree [18].

In order to capture the restrictions on the automorphism group, we introduce the notion of (t, k) -WL-bounded graphs which generalize so-called t -CR-bounded graphs. The class of t -CR-bounded graphs was originally defined by Ponomarenko [23] and was recently rediscovered in [21, 10, 22] in a series of works eventually leading to an algorithm testing isomorphism of graphs excluding K_h as a topological subgraph in time $n^{\mathcal{O}((\log h)^c)}$. Intuitively speaking, a graph G is t -CR-bounded if an initially uniform vertex-coloring χ can be turned into a discrete coloring (i.e., a coloring where every vertex has its own color) by repeatedly (a) applying the standard Color Refinement algorithm, and (b) splitting all color classes of size at most t . We define (t, k) -WL-bounded graphs in the same way, but replace the Color Refinement algorithm by the well-known Weisfeiler-Leman algorithm of dimension k (see, e.g., [5, 12]). Maybe surprisingly, this natural extension of t -CR-bounded has not been considered so far in the literature, and we start by building a polynomial-time isomorphism test for such graphs using the group-theoretic methods developed by Luks [18] as well as a simple extension due to Miller [20]. Actually, it turns out that isomorphism of (t, k) -WL-bounded graphs can even be tested in time $n^{\mathcal{O}(k \cdot (\log t)^c)}$ using recent extensions [21] of Babai's quasipolynomial-time isomorphism test. However, since we only apply these methods for $t = k = 2$, there is no need for our algorithm to rely on such sophisticated subroutines.

Now, as the main structural insight, we prove that each 3-connected graph G that excludes $K_{3,h}$ as a minor admits (after fixing 3 vertices) an isomorphism-invariant rooted tree decomposition (T, β) such that the adhesion width (i.e., the maximal intersection between two bags) is bounded by h . Additionally, each bag $\beta(t)$, $t \in V(T)$, can be equipped with a set $\gamma(t) \subseteq \beta(t)$ of size $|\gamma(t)| \leq h^4$ such that, after fixing all vertices in $\gamma(t)$, G restricted to $\beta(t)$ is $(2, 2)$ -WL-bounded. Given such a decomposition, isomorphisms can be computed by a simple bottom-up dynamic programming strategy along the tree decompositions. For each bag, isomorphism is tested by first individualizing all vertices from $\gamma(t)$ at an additional factor of $|\gamma(t)|! = 2^{\mathcal{O}(h^4 \log h)}$ in the running time. Following the individualization of these vertices, our algorithm can then simply rely on a polynomial-time isomorphism test for $(2, 2)$ -WL-bounded graphs. Here, we incorporate the partial solutions computed in the subtree below the current bag via a simple gadget construction.

To compute the decomposition (T, β) , we also build on the notion of $(2, 2)$ -WL-bounded graphs. Given a set $X \subseteq V(G)$, we define the $(2, 2)$ -closure to be the set $D = \text{cl}_{2,2}^G(X)$ of all vertices appearing in a singleton color class after artificially individualizing all vertices from X , and performing the $(2, 2)$ -WL procedure. As one of the main technical contributions, we can show that the interplay between D and its complement in G is simple (assuming G excludes $K_{3,h}$ as a minor). To be more precise, building on various properties of the 2-dimensional Weisfeiler-Leman algorithm, we show that $|N_G(Z)| < h$ for every connected component Z of $G - D$. This allows us to choose $D = \text{cl}_{2,2}^G(X)$ as the root bag of (T, β) for some carefully chosen set X , and obtain the decomposition (T, β) by recursion.

2 Preliminaries

2.1 Graphs

A *graph* is a pair $G = (V(G), E(G))$ consisting of a *vertex set* $V(G)$ and an *edge set* $E(G)$. All graphs considered in this paper are finite and simple (i.e., they contain no loops or multiple edges). Moreover, unless explicitly stated otherwise, all graphs are undirected. For an undirected graph G and $v, w \in V(G)$, we write vw as a shorthand for $\{v, w\} \in E(G)$. The *neighborhood* of a vertex $v \in V(G)$ is denoted by $N_G(v)$. The *degree* of v , denoted by $\deg_G(v)$, is the number of edges incident with v , i.e., $\deg_G(v) = |N_G(v)|$. For $X \subseteq V(G)$, we define $N_G(X) := (\bigcup_{v \in X} N_G(v)) \setminus X$. If the graph G is clear from context, we usually omit the index and simply write $N(v)$, $\deg(v)$ and $N(X)$. We write $K_{\ell,h}$ to denote the complete bipartite graph on ℓ vertices on the left side and h vertices on the right side. For two sets $A, B \subseteq V(G)$, we denote by $E_G(A, B) := \{vw \in E(G) \mid v \in A, w \in B\}$. Also, $G[A, B]$ denotes the graph with vertex set $A \cup B$ and edge set $E_G(A, B)$. Moreover, $G[A] := G[A, A]$ denotes the induced subgraph on A , and $G - A$ the subgraph induced by the complement of A , that is, the graph $G - A := G[V(G) \setminus A]$. For $F \subseteq E(G)$, we also define $G - F$ to be the graph obtained from G by removing all edges contained in F (the vertex set remains unchanged). A graph H is a *subgraph* of G , denoted by $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A graph H is a *minor* of G if H can be obtained from G by deleting vertices and edges, as well as contracting edges. The graph G *excludes* H *as a minor* if it does not have a minor isomorphic to H .

An *isomorphism* from G to a graph H is a bijection $\varphi: V(G) \rightarrow V(H)$ that respects the edge relation, that is, for all $v, w \in V(G)$, it holds that $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$. Two graphs G and H are *isomorphic*, written $G \cong H$, if there is an isomorphism from G to H . We write $\varphi: G \cong H$ to denote that φ is an isomorphism

from G to H . Also, $\text{Iso}(G, H)$ denotes the set of all isomorphisms from G to H . The automorphism group of G is $\text{Aut}(G) := \text{Iso}(G, G)$. Observe that, if $\text{Iso}(G, H) \neq \emptyset$, it holds that $\text{Iso}(G, H) = \text{Aut}(G)\varphi := \{\gamma\varphi \mid \gamma \in \text{Aut}(G)\}$ for every isomorphism $\varphi \in \text{Iso}(G, H)$.

A *vertex-colored graph* is a tuple (G, χ_V) where G is a graph and $\chi_V: V(G) \rightarrow C$ is a mapping into some set C of colors, called *vertex-coloring*. Similarly, an *arc-colored graph* is a tuple (G, χ_E) , where G is a graph and $\chi_E: \{(u, v) \mid \{u, v\} \in E(G)\} \rightarrow C$ is a mapping into some color set C , called *arc-coloring*. Observe that colors are assigned to directed edges, i.e., the directed edge (v, w) may obtain a different color than (w, v) . We also consider vertex- and arc-colored graphs (G, χ_V, χ_E) where χ_V is a vertex-coloring and χ_E is an arc-coloring. Typically, C is chosen to be an initial segment $[n] := \{1, \dots, n\}$ of the natural numbers. To be more precise, we generally assume that there is a linear order on the set of all potential colors which, for example, allows us to identify a minimal color appearing in a graph in a unique way. Isomorphisms between vertex- and arc-colored graphs have to respect the colors of the vertices and arcs.

2.2 Weisfeiler-Leman Algorithm

The Weisfeiler-Leman algorithm, originally introduced by Weisfeiler and Leman in its 2-dimensional version [28], forms one of the most fundamental subroutines in the context of isomorphism testing.

Let $\chi_1, \chi_2: V^k \rightarrow C$ be colorings of k -tuples, where C is a finite set of colors. We say χ_1 *refines* χ_2 , denoted $\chi_1 \preceq \chi_2$, if $\chi_1(\bar{v}) = \chi_1(\bar{w})$ implies $\chi_2(\bar{v}) = \chi_2(\bar{w})$ for all $\bar{v}, \bar{w} \in V^k$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

We describe the *k-dimensional Weisfeiler-Leman algorithm* (k -WL) for all $k \geq 1$. For an input graph G let $\chi_{(0)}^k[G]: (V(G))^k \rightarrow C$ be the coloring where each tuple is colored with the isomorphism type of its underlying ordered subgraph. More precisely, $\chi_{(0)}^k[G](v_1, \dots, v_k) = \chi_{(0)}^k[G](v'_1, \dots, v'_k)$ if and only if, for all $i, j \in [k]$, it holds that $v_i = v_j \Leftrightarrow v'_i = v'_j$ and $v_i v_j \in E(G) \Leftrightarrow v'_i v'_j \in E(G)$. If the graph is equipped with a coloring the initial coloring $\chi_{(0)}^k[G]$ also takes the input coloring into account. More precisely, for a vertex-coloring χ_V , it additionally holds that $\chi_V(v_i) = \chi_V(v'_i)$ for all $i \in [k]$. And for an arc-coloring χ_E , it is the case that $\chi_E(v_i, v_j) = \chi_E(v'_i, v'_j)$ for all $i, j \in [k]$ such that $v_i v_j \in E(G)$.

We then recursively define the coloring $\chi_{(i)}^k[G]$ obtained after i rounds of the algorithm. For $k \geq 2$ and $\bar{v} = (v_1, \dots, v_k) \in (V(G))^k$ let $\chi_{(i+1)}^k[G](\bar{v}) := \left(\chi_{(i)}^k[G](\bar{v}), \mathcal{M}_i(\bar{v}) \right)$ where

$$\mathcal{M}_i(\bar{v}) := \left\{ \left(\chi_{(i)}^k[G](\bar{v}[w/1]), \dots, \chi_{(i)}^k[G](\bar{v}[w/k]) \right) \mid w \in V(G) \right\}$$

and $\bar{v}[w/i] := (v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_k)$ is the tuple obtained from \bar{v} by replacing the i -th entry by w (and $\{\dots\}$ denotes a multiset). For $k = 1$ the definition is similar, but we only iterate over neighbors of v , i.e., $\chi_{(i+1)}^1[G](v) := \left(\chi_{(i)}^1[G](v), \mathcal{M}_i(v) \right)$ where

$$\mathcal{M}_i(v) := \left\{ \chi_{(i)}^1[G](w) \mid w \in N_G(v) \right\}.$$

By definition, $\chi_{(i+1)}^k[G] \preceq \chi_{(i)}^k[G]$ for all $i \geq 0$. Hence, there is a minimal i_∞ such that $\chi_{(i_\infty)}^k[G] \equiv \chi_{(i_\infty+1)}^k[G]$ and for this i_∞ the coloring $\chi_{\text{WL}}^k[G] := \chi_{(i_\infty)}^k[G]$ is the *k-stable* coloring of G . The k -dimensional Weisfeiler-Leman algorithm takes as input a (vertex- or arc-)colored graph G and returns (a coloring that is equivalent to) $\chi_{\text{WL}}^k[G]$. This can be implemented in time $\mathcal{O}(n^{k+1} \log n)$ (see [12]).

2.3 Group Theory

We introduce the group-theoretic notions required in this work. We refer to [26, 6] for further background.

Permutation groups. A *permutation group* acting on a set Ω is a subgroup $\Gamma \leq \text{Sym}(\Omega)$ of the symmetric group. The size of the permutation domain Ω is called the *degree* of Γ . If $\Omega = [n]$, then we also write S_n instead of $\text{Sym}(\Omega)$. For $\gamma \in \Gamma$ and $\alpha \in \Omega$ we denote by α^γ the image of α under the permutation γ . For $A \subseteq \Omega$ and $\gamma \in \Gamma$ let $A^\gamma := \{\alpha^\gamma \mid \alpha \in A\}$. The set A is Γ -*invariant* if $A^\gamma = A$ for all $\gamma \in \Gamma$. For a partition \mathcal{P} of Ω let $\mathcal{P}^\gamma := \{A^\gamma \mid A \in \mathcal{P}\}$. Observe that \mathcal{P}^γ is again a partition of Ω . We say \mathcal{P} is Γ -*invariant* if $\mathcal{P}^\gamma = \mathcal{P}$ for all $\gamma \in \Gamma$.

For $A \subseteq \Omega$ and a bijection $\theta: \Omega \rightarrow \Omega'$ we denote by $\theta[A]$ the restriction of θ to the domain A . For a Γ -invariant set $A \subseteq \Omega$, we denote by $\Gamma[A] := \{\gamma[A] \mid \gamma \in \Gamma\}$ the induced action of Γ on A , i.e., the group obtained from Γ by restricting all permutations to A . More generally, for every set Λ of bijections with domain Ω , we denote by $\Lambda[A] := \{\theta[A] \mid \theta \in \Lambda\}$. Similarly, for a partition \mathcal{P} of Ω , we denote by $\theta[\mathcal{P}]: \mathcal{P} \rightarrow \mathcal{P}'$ the mapping defined via $\theta(A) := \{\theta(\alpha) \mid \alpha \in A\}$ for all $A \in \mathcal{P}$. As before, $\Lambda[\mathcal{P}] := \{\theta[\mathcal{P}] \mid \theta \in \Lambda\}$.

Groups with restricted composition factors. We shall be interested in a particular subclass of permutation groups, namely groups with restricted composition factors. Let Γ be a group. A *subnormal series* is a sequence of subgroups $\Gamma = \Gamma_0 \geq \Gamma_1 \geq \dots \geq \Gamma_k = \{\text{id}\}$ such that Γ_i is a normal subgroup of Γ_{i-1} for all $i \in [k]$. The length of the series is k and the groups Γ_{i-1}/Γ_i are the factor groups of the series, $i \in [k]$. A *composition series* is a strictly decreasing subnormal series of maximal length. For every finite group Γ all composition series have the same family (considered as a multiset) of factor groups (cf. [26]). A *composition factor* of a finite group Γ is a factor group of a composition series of Γ .

► **Definition 2.** For $d \geq 2$ let $\widehat{\Gamma}_d$ denote the class of all groups Γ for which every composition factor of Γ is isomorphic to a subgroup of S_d . A group Γ is a $\widehat{\Gamma}_d$ -group if it is contained in the class $\widehat{\Gamma}_d$.

Let us point out the fact that there are two similar classes of groups usually referred by Γ_d in the literature. The first is the class denoted by $\widehat{\Gamma}_d$ here originally introduced by Luks [18], while the second one, for example used in [2], in particular allows composition factors that are simple groups of Lie type of bounded dimension.

Group-Theoretic Tools for Isomorphism Testing. In this work, the central group-theoretic subroutine is an isomorphism test for hypergraphs where the input group is a $\widehat{\Gamma}_d$ -group. Two hypergraphs $\mathcal{H}_1 = (V_1, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V_2, \mathcal{E}_2)$ are isomorphic if there is a bijection $\varphi: V_1 \rightarrow V_2$ such that $E \in \mathcal{E}_1$ if and only if $E^\varphi \in \mathcal{E}_2$ for all $E \in 2^{V_1}$ (where $E^\varphi := \{\varphi(v) \mid v \in E\}$ and 2^{V_1} denotes the power set of V_1). We write $\varphi: \mathcal{H}_1 \cong \mathcal{H}_2$ to denote that φ is an isomorphism from \mathcal{H}_1 to \mathcal{H}_2 . Consistent with previous notation, we denote by $\text{Iso}(\mathcal{H}_1, \mathcal{H}_2)$ the set of isomorphisms from \mathcal{H}_1 to \mathcal{H}_2 . More generally, for $\Gamma \leq \text{Sym}(V_1)$ and a bijection $\theta: V_1 \rightarrow V_2$, we define $\text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2) := \{\varphi \in \Gamma\theta \mid \varphi: \mathcal{H}_1 \cong \mathcal{H}_2\}$. In this work, we define the Hypergraph Isomorphism Problem to take as input two hypergraphs $\mathcal{H}_1 = (V_1, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V_2, \mathcal{E}_2)$, a group $\Gamma \leq \text{Sym}(V_1)$ and a bijection $\theta: V_1 \rightarrow V_2$, and the goal is to compute a representation² of $\text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2)$. The following algorithm forms a crucial subroutine.

² While $\text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2)$ may be exponentially large, it can be represented by a single isomorphism $\varphi \in \text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2)$ and a generating set for $\text{Aut}_\Gamma(\mathcal{H}_1) := \text{Iso}_\Gamma(\mathcal{H}_1, \mathcal{H}_1)$. For general background on how to perform computations on permutation groups, I refer to [27].

► **Theorem 3** (Miller [20]). *Let $\mathcal{H}_1 = (V_1, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V_2, \mathcal{E}_2)$ be two hypergraphs and let $\Gamma \leq \text{Sym}(V_1)$ be a $\widehat{\Gamma}_d$ -group and $\theta: V_1 \rightarrow V_2$ a bijection. Then $\text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2)$ can be computed in time $(n + m)^{\mathcal{O}(d)}$ where $n := |V_1|$ and $m := |\mathcal{E}_1|$.*

► **Theorem 4** (Neuen [21]). *Let $\mathcal{H}_1 = (V_1, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V_2, \mathcal{E}_2)$ be two hypergraphs and let $\Gamma \leq \text{Sym}(V_1)$ be a $\widehat{\Gamma}_d$ -group and $\theta: V_1 \rightarrow V_2$ a bijection. Then $\text{Iso}_{\Gamma\theta}(\mathcal{H}_1, \mathcal{H}_2)$ can be computed in time $(n + m)^{\mathcal{O}((\log d)^c)}$ for some constant c where $n := |V_1|$ and $m := |\mathcal{E}_1|$.*

Observe that both algorithms given by the two theorems tackle the same problem. The second algorithm is asymptotically much faster, but it is also much more complicated and the constant factors in the exponent of the running time are likely to be much higher. Since this paper only applies either theorem for $d = 2$, it seems to be preferable to use the first algorithm. Indeed, the first result is a simple extension of Luks’s well-known isomorphism test for bounded-degree graphs [18], and thus the underlying algorithm is fairly simple. For all these reasons, we mostly build on Theorem 3. However, for future applications of the techniques presented in this work, it might be necessary to build on Theorem 4 to benefit from the improved run time bound. For this reason, we shall provide variants of our results building on Theorem 4 wherever appropriate.

3 Allowing Weisfeiler and Leman to Split Small Color Classes

In this section, we introduce the concept of (t, k) -WL-bounded graphs and provide a polynomial-time isomorphism test for such graphs for all constant values of t and k . The final fpt isomorphism test for graphs excluding $K_{3,h}$ as a minor builds on this subroutine for $t = k = 2$.

The concept of (t, k) -WL-bounded graphs is a natural extension of t -CR-bounded graphs which were already introduced by Ponomarenko in the late 1980’s [23] and which were recently rediscovered in [21, 10, 22]. Intuitively speaking, a graph G is t -CR-bounded, $t \in \mathbb{N}$, if an initially uniform vertex-coloring χ (i.e., all vertices receive the same color) can be turned into the discrete coloring (i.e., each vertex has its own color) by repeatedly

- performing the Color Refinement algorithm (expressed by the letters “CR”), and
- taking a color class $[v]_\chi := \{w \in V(G) \mid \chi(w) = \chi(v)\}$ of size $|[v]_\chi| \leq t$ and assigning each vertex from the class its own color.

A very natural extension of this idea to replace the Color Refinement algorithm by the Weisfeiler-Leman algorithm for some fixed dimension k . This leads us to the notion of (t, k) -WL-bounded graphs (the letters “CR” are replaced by “ k -WL”). In particular, $(t, 1)$ -WL-bounded graphs are exactly the t -CR-bounded graphs. Maybe surprisingly, it seems that this simple extension has not been considered so far in the literature.

► **Definition 5.** *A vertex- and arc-colored graph $G = (V, E, \chi_V, \chi_E)$ is (t, k) -WL-bounded if the sequence $(\chi_i)_{i \geq 0}$ reaches a discrete coloring where $\chi_0 := \chi_V$,*

$$\chi_{2i+1}(v) := \chi_{\text{WL}}^k[V, E, \chi_{2i}, \chi_E](v, \dots, v)$$

and

$$\chi_{2i+2}(v) := \begin{cases} (v, 1) & \text{if } |[v]_{\chi_{2i+1}}| \leq t \\ (\chi_{2i+1}(v), 0) & \text{otherwise} \end{cases}$$

for all $i \geq 0$.

Also, for the minimal $i_\infty \geq 0$ such that $\chi_{i_\infty} \equiv \chi_{i_\infty+1}$, we refer to χ_{i_∞} as the (t, k) -WL-stable coloring of G and denote it by $\chi_{(t,k)\text{-WL}}[G]$.

At this point, the reader may wonder why $(\chi_i)_{i \geq 0}$ is chosen as a sequence of vertex-colorings and not a sequence of colorings of k -tuples of vertices (since k -WL also colors k -tuples of vertices). While such a variant certainly makes sense, it still leads to the same class of graphs. Let G be a graph and let $\chi := \chi_{\text{WL}}^k[G]$. The main insight is that, if there is some color $c \in \text{im}(\chi)$ for which $|\chi^{-1}(c)| \leq t$, then there is also a color $c' \in \text{im}(\chi)$ for which $|\chi^{-1}(c')| \leq t$ and $\chi^{-1}(c') \subseteq \{(v, \dots, v) \mid v \in V(G)\}$. In other words, one can not achieve any additional splitting of color classes by also considering non-diagonal color classes.

We also need to extend several notions related to t -CR-bounded graphs. Let G be a graph and let $X \subseteq V(G)$ be a set of vertices. Let $\chi_V^*: V(G) \rightarrow C$ be the vertex-coloring obtained from individualizing all vertices in the set X , i.e., $\chi_V^*(v) := (v, 1)$ for $v \in X$ and $\chi_V^*(v) := (0, 0)$ for $v \in V(G) \setminus X$. Let $\chi := \chi_{(t,k)\text{-WL}}[G, \chi_V^*]$ denote the (t, k) -WL-stable coloring with respect to the input graph (G, χ_V^*) . We define the (t, k) -closure of the set X (with respect to G) to be the set

$$\text{cl}_{t,k}^G(X) := \{v \in V(G) \mid |[v]_\chi| = 1\}.$$

Observe that $X \subseteq \text{cl}_{t,k}^G(X)$. For $v_1, \dots, v_\ell \in V(G)$ we also use $\text{cl}_{t,k}^G(v_1, \dots, v_\ell)$ as a shorthand for $\text{cl}_{t,k}^G(\{v_1, \dots, v_\ell\})$. If the input graph is equipped with a vertex- or arc-coloring, all definitions are extended in the natural way.

Now, we concern ourselves with designing a polynomial-time isomorphism test for (t, k) -WL-bounded graphs. Actually, we shall prove a slightly stronger result which turns out to be useful later on. The main idea for the algorithm is to build a reduction to the isomorphism problem for $(t, 1)$ -WL-bounded graphs for which such results are already known [23, 21]. Indeed, isomorphism of $(t, 1)$ -WL-bounded graphs can be reduced to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$ -groups. Since one may be interested in using different subroutines for solving the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$ -groups (see the discussion at the end of Section 2.3), the main result is stated via an oracle for the Hypergraph Isomorphism Problem on $\widehat{\Gamma}_t$ -groups.

► **Theorem 6.** *Let G_1, G_2 be two vertex- and arc-colored graphs and let $\chi_i := \chi_{(t,k)\text{-WL}}[G_i]$. Also let $\mathcal{P}_i = \{[v]_{\chi_i} \mid v \in V(G_i)\}$ be the partition into color classes of χ_i . Then $\mathcal{P}_1^\varphi = \mathcal{P}_2$ for all $\varphi \in \text{Iso}(G_1, G_2)$.*

Moreover, using oracle access to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$ -groups, in time $n^{\mathcal{O}(k)}$ one can compute a $\widehat{\Gamma}_t$ -group $\Gamma \leq \text{Sym}(\mathcal{P}_1)$ and a bijection $\theta: \mathcal{P}_1 \rightarrow \mathcal{P}_2$ such that

$$\text{Iso}(G_1, G_2)[\mathcal{P}_1] \subseteq \Gamma\theta.$$

In particular, $\text{Aut}(G_1)[\mathcal{P}_1] \in \widehat{\Gamma}_t$.

► **Corollary 7.** *Let G_1, G_2 be two (t, k) -WL-bounded graphs. Then a representation for $\text{Iso}(G_1, G_2)$ can be computed in time $n^{\mathcal{O}(k \cdot (\log t)^c)}$ for some absolute constant c .*

4 Structure Theory and Small Color Classes

Having established the necessary tools, we can now turn to the isomorphism test for graphs excluding $K_{3,h}$ as a minor. We start by giving a high-level overview on the algorithm. The main idea is to build on the isomorphism test for $(2, 2)$ -WL-bounded graphs described in the last section. Let G_1 and G_2 be two (vertex- and arc-colored) graphs that exclude $K_{3,h}$ as a minor. Using well-known reduction techniques building on isomorphism-invariant decompositions into triconnected³ components (see, e.g., [11]), we may assume without loss of generality that G_1 and G_2 are 3-connected.

³ A triconnected component is either 3-connected or a cycle.

The algorithm starts by individualizing three vertices. To be more precise, the algorithm picks three distinct vertices $v_1, v_2, v_3 \in V(G_1)$ and iterates over all choices of potential images $w_1, w_2, w_3 \in V(G_2)$ under some isomorphism between G_1 and G_2 . Let $X_1 := \{v_1, v_2, v_3\}$ and $X_2 := \{w_1, w_2, w_3\}$. Also, let $D_i := \text{cl}_{2,2}^{G_i}(X_i)$ denote the $(2, 2)$ -closure of X_i , $i \in \{1, 2\}$. Observe that D_i is defined in an isomorphism-invariant manner given the initial choice of X_i . Building on Theorems 3 and 6 it can be checked in polynomial time whether G_1 and G_2 are isomorphic restricted to the sets D_1 and D_2 .

Now, the central idea is to follow a decomposition strategy. Let Z_1^i, \dots, Z_ℓ^i denote the vertex sets of the connected components of $G_i - D_i$, and let $S_j^i := N_{G_i}(Z_j^i)$ for $j \in [\ell]$ and $i \in \{1, 2\}$. We recursively compute isomorphisms between all pairs of graphs $G_i[Z_j^i \cup S_j^i]$ for all $j \in [\ell]$ and $i \in \{1, 2\}$. To be able to determine whether all these partial isomorphisms can be combined into a global isomorphism, the crucial insight is that $|S_j^i| < h$ for all $j \in [\ell]$ and $i \in \{1, 2\}$.

► **Lemma 8.** *Let G be a graph that excludes $K_{3,h}$ as a minor. Also let $X \subseteq V(G)$ and define $D := \text{cl}_{2,2}^G(X)$. Let Z be a connected component of $G - D$. Then $|N_G(Z)| < h$.*

Indeed, this lemma forms one of the main technical contributions of the paper. I remark that similar statements are exploited in [10, 21, 22] eventually leading to an isomorphism test running in time $n^{\mathcal{O}((\log h)^c)}$ for all graphs excluding K_h as a topological subgraph. However, all these variants require the (t, k) -closure to be taken for non-constant values of t (i.e., $t = \Omega(h)$). For the design of an fpt-algorithm, this is infeasible since we can only afford to apply Theorem 6 for constant values of t and k (since D_i might be equal to $V(G_i)$).

The lemma above implies that the interplay between D_i and $V(G_i) \setminus D_i$ is simple which allows for a dynamic programming approach. To be more precise, we can recursively list all elements of the set $\text{Iso}((G_i[Z_j^i \cup S_j^i], S_j^i), (G_{i'}[Z_{j'}^{i'} \cup S_{j'}^{i'}], S_{j'}^{i'}))[S_j^i]$ for all $j, j' \in [\ell]$ and $i, i' \in \{1, 2\}$ (i.e., we list all bijections $\sigma: S_j^i \rightarrow S_{j'}^{i'}$ that can be extended to an isomorphism between the corresponding subgraphs). To incorporate this information, we extend the graph $G_i[D_i]$ by simple gadgets obtaining graphs H_i that are $(2, 2)$ -WL-bounded and such that $G_1 \cong G_2$ if and only if $H_1 \cong H_2$. (For technical reasons, the algorithm does not exactly implement this strategy, but closely follows the general idea.)

In order to realize this recursive strategy, it remains to ensure that the algorithm makes progress when performing a recursive call. Actually, this turns out to be a non-trivial task. Indeed, it may happen that $D_i = X_i$, there is only a single component Z_1^i of $G_i - D_i$, and $N_{G_i}(Z_1^i) = D_i$. To circumvent this problem, the idea is to compute an isomorphism-invariant extension $\gamma(X_i) \supseteq X_i$ such that $|\gamma(X_i)| \leq h^4$. Assuming such an extension can be computed, we simply extend the set X_i until the algorithm arrives in a situation where the recursive scheme discussed above makes progress. Observe that this is guaranteed to happen as soon as $|X_i| \geq h$ building on Lemma 8. Also note that we can still artificially individualize all vertices from X_i at a cost of $2^{\mathcal{O}(h^4 \log h)}$ (since any isomorphism can only map vertices from X_1 to vertices from X_2).

To compute the extension, we exploit the fact that G_i is $(h-1, 1)$ -WL-bounded by [21, Corollary 24] (after individualizing 3 vertices). Simply speaking, for every choice of three distinct vertices in X_i , after individualizing these vertices and performing the 1-dimensional Weisfeiler-Leman algorithm, we can identify a color class of size at most $h-1$ to be added to the set X_i . Overall, assuming $|X_i| \leq h$, this gives an extension $\gamma(X_i)$ of size at most $h + h^3(h-1) \leq h^4$.

This completes the description of the general strategy. In the following sections, we provide more detailed arguments. We first provide a sketch on the proof of Lemma 8 in the next section. Afterwards, we compute the entire decompositions of the input graphs in Section 6. Finally, the dynamic programming strategy along the computed decompositions is implemented in Section 7.

5 Finding Disjoint and Connected Subgraphs

In this section, we give some details on the proof of Lemma 8. Let us start by introducing some additional notation for the 2-dimensional Weisfeiler-Leman algorithm.

Let G be a graph and let $\chi := \chi_{\text{WL}}^2[G]$ be the coloring computed by the 2-dimensional Weisfeiler-Leman algorithm. We denote by $C_V = C_V(G, \chi) := \{\chi(v, v) \mid v \in V(G)\}$ the set of *vertex colors* under the coloring χ . Also, for $c \in C_V$, $V_c := \{v \in V(G) \mid \chi(v, v) = c\}$ denotes the set of all vertices of color c . Moreover, we define the graph $G[[\chi]]$ with vertex set $V(G[[\chi]]) := C_V(G, \chi)$ and edges $E(G[[\chi]]) := \{c_1 c_2 \mid \exists v_1 \in V_{c_1}, v_2 \in V_{c_2} : v_1 v_2 \in E(G)\}$.

The next lemma builds the main technical step in the proof of Lemma 8.

► **Lemma 9.** *Let G be a graph and let χ be a 2-stable coloring. Suppose that $G[[\chi]]$ is connected and $|V_c| \geq 3$ for every $c \in C_V$. Then there are vertex-disjoint, connected subgraphs $H_1, H_2, H_3 \subseteq G$ such that $V(H_r) \cap V_c \neq \emptyset$ for all $r \in \{1, 2, 3\}$ and $c \in C_V$.*

Proof Idea. Let F be a spanning tree of $G[[\chi]]$ and fix an arbitrary root node $c_0 \in V(F) = C_V$. On a high-level, the graphs H_1, H_2, H_3 are constructed in a top-to-bottom fashion along the tree F . To start, let us select three arbitrary distinct vertices $v_1, v_2, v_3 \in V_{c_0}$ and add v_r to the graph H_r . Now, let d be a color which is already covered (i.e., $V_d \cap H_r \neq \emptyset$ for all $r \in \{1, 2, 3\}$), and let c be a child of d which is not covered. Consider the graph $G[V_d, V_c]$. Let $v_r \in V_d \cap V(H_r)$ for $r \in \{1, 2, 3\}$. If there are vertices $w_1, w_2, w_3 \in V_c$ such that $v_r w_r \in E(G)$ then we can simply add vertex w_r as well as the edge $v_r w_r$ to H_r in order to cover the color class V_c . Assuming we can always find such vertices, this strategy can be repeated going down the tree F until all color classes are covered.

So suppose that there are no such vertices w_1, w_2, w_3 . By Hall's Marriage Theorem, this means there is a set $V' \subseteq \{v_1, v_2, v_3\}$ such that $|N(V') \cap V_c| < |V'|$. Since χ is 2-stable, we get that $|N(v) \cap V_c| = |N(v') \cap V_c|$ for all $v, v' \in V_d$. Together, this means there is some $\delta \in \{1, 2\}$ such that $|N(v) \cap V_c| = \delta$ for all $v \in V_d$.

First, suppose that $\delta = 1$. Then $G[V_d, V_c]$ is isomorphic to a disjoint union of ℓ stars $K_{1,h}$, for some $\ell \geq 3$ and $h \geq 2$. In this situation, it is possible to contract the connected components of $G[V_d, V_c]$ to single vertices, and proceed by induction. At this point, we crucially exploit that using the 2-dimensional Weisfeiler-Leman algorithm allows us to show that all color classes in the contracted graph still have size at least 3 (such a statement is not true when using the Color Refinement algorithm). By induction, we obtain graphs H'_1, H'_2, H'_3 . To obtain the original graphs, we simply uncontract any contracted vertices contained in H'_1, H'_2, H'_3 .

In the other case, we have $\delta = 2$. Let us partition V_d into *c-twin-classes* where vertices $v, v' \in V_d$ are declared to be *c-twins* if $N(v) \cap V_c = N(v') \cap V_c$. If there are at least 3 twin-classes, then it is again possible to contract the twin-classes to single vertices and proceed by induction. Here, the crucial observation is that the *c-twin-classes* are non-trivial since $|N(V') \cap V_c| < |V'|$ for some set $V' \subseteq \{v_1, v_2, v_3\}$. The critical case occurs if there are exactly 2 twin-classes meaning that $G[V_d, V_c]$ is isomorphic to a disjoint union of 2 copies of $K_{2,h}$, for some $h \geq 3$ (in this case $|V_c| = 4$). Now, the basic idea is to ensure that v_1, v_2, v_3 cover

both connected components (which means there are vertices w_1, w_2, w_3 as above). However, this additional requirement comes with severe additional complications. First, information of this type needs to be propagated up the tree (i.e., vertices in the root color class may already need to be chosen appropriately to avoid problematic situations later on). But much more problematically, each child of c may add a different restriction which all need to be met at the same time. Here, we again crucially rely on the 2-dimensional Weisfeiler-Leman algorithm to show that all requirements can indeed be met at the same time. Unfortunately, this comes at the price that each vertex of V_d has to be contained in one of the graphs H_r , $r \in \{1, 2, 3\}$ (this allows us to choose different triples (v_1, v_2, v_3) for different children of d). To ensure that H_r remains connected, we introduce a second type of restriction that is passed down the tree, and which ensures that all vertices from V_d , which are added to H_r , end up in the same connected component of H_r . By carefully implementing the induction, it can be shown that all these additional requirements can indeed be realized. \blacktriangleleft

Proof of Lemma 8. Let χ be a 2-stable coloring such that $|\chi[v]| = 1$ for all $v \in D$ and $|\chi[w]| \geq 3$ for all $w \in V(G) \setminus D$. Suppose towards a contradiction that $|N_G(Z)| \geq h$, and pick $v_1, \dots, v_h \in N_G(Z)$ to be distinct vertices. Let $C := \{\chi(v, v) \mid v \in Z\}$ be the set of vertex colors appearing in the set Z . Note that $(G[\chi])[C]$ is connected, and $|V_c| \geq 3$ for all $c \in C$. Let $W := \{w \in V(G) \mid \chi(w, w) \in C\}$. Observe that $W \cap D = \emptyset$. By Lemma 9, there are connected, vertex-disjoint subgraphs $H_1, H_2, H_3 \subseteq G[W]$ such that $V(H_r) \cap V_c \neq \emptyset$ for all $r \in \{1, 2, 3\}$ and $c \in C$.

Now let $i \in [h]$. Since $v_i \in N_G(Z)$ there is some vertex $w_i \in Z \subseteq W$ such that $v_i w_i \in E(G)$. Let $c_i := \chi(w_i, w_i)$. Observe that $c_i \in C$. Also, $V_{c_i} \subseteq N_G(v_i)$ since $|\chi[v_i]| = 1$ and χ is 2-stable. This implies that $N_G(v_i) \cap V(H_r) \neq \emptyset$ for all $r \in \{1, 2, 3\}$, because $V(H_r) \cap V_{c_i} \neq \emptyset$. But this results in a minor isomorphic to $K_{3,h}$ with vertices v_1, \dots, v_h on the right side, and vertices $V(H_1), V(H_2), V(H_3)$ on the left side. \blacktriangleleft

Besides Lemma 8, we also require a second tool which is used to define the extension sets $\gamma(X_i)$ which we needed to ensure the recursive algorithm makes progress.

► Lemma 10. *Let G be a graph that excludes $K_{3,h}$ as a minor. Also let $X \subseteq V(G)$ and define $D := \text{cl}_{h-1,1}^G(X)$. Let Z be a connected component of $G - D$. Then $|N_G(Z)| < 3$.*

The lemma essentially follows from [21, Lemma 23]. For the sake of completeness and due to its simplicity, a complete proof is still given below.

Proof. Let χ be a 1-stable coloring such that $|\chi[v]| = 1$ for all $v \in D$ and $|\chi[w]| \geq h$ for all $w \in V(G) \setminus D$. Suppose towards a contradiction that $|N_G(Z)| \geq 3$, and pick $v_1, v_2, v_3 \in N_G(Z)$ to be distinct vertices. Let $C := \{\chi(v) \mid v \in Z\}$, and define H to be the graph with $V(H) := C$ and

$$E(H) := \{c_1 c_2 \mid \exists v_1 \in \chi^{-1}(c_1), v_2 \in \chi^{-1}(c_2): v_1 v_2 \in E(G)\}.$$

Let T be a spanning tree of H . Also, for each $i \in \{1, 2, 3\}$, fix a color $c_i \in C$ such that $N_G(v_i) \cap \chi^{-1}(c_i) \neq \emptyset$. Let T' be the induced subtree obtained from T by repeatedly removing all leaves distinct from c_1, c_2, c_3 . Finally, let T'' be the tree obtained from T' by adding three fresh vertices v_1, v_2, v_3 where v_i is connected to c_i . Observe that v_1, v_2, v_3 are precisely the leaves of T'' . Now, T'' contains a unique node c of degree three (possibly $c = c_i$ for some $i \in \{1, 2, 3\}$). Observe that $|\chi^{-1}(c)| \geq h$. We define C_i to be the set of all internal vertices which appear on the unique path from v_i to c in the tree T'' . Finally, define $U_i := \{v_i\} \cup \bigcup_{c' \in C_i} \chi^{-1}(c')$.

Since χ is 1-stable and $|[v_i]_\chi| = 1$ we get that $G[U_i]$ is connected for all $i \in \{1, 2, 3\}$. Also, $E_G(U_i, \{w\}) \neq \emptyset$ for all $w \in \chi^{-1}(c)$ and $i \in \{1, 2, 3\}$. But this provides a minor isomorphic to $K_{3,h}$ with vertices U_1, U_2, U_3 on the left side and the vertices from $\chi^{-1}(c)$ on the right side. \blacktriangleleft

6 A Decomposition Theorem

Next, we use the insights gained in the last section to prove a decomposition theorem for graphs that exclude $K_{3,h}$ as a minor. In the following, all tree decompositions are rooted, i.e., there is a designated root node and we generally assume all edges to be directed away from the root.

► **Theorem 11.** *Let $h \geq 3$. Let G be a 3-connected graph, and suppose $S \subseteq V(G)$ such that*

(A) $G - E(S, S)$ excludes $K_{3,h}$ as a minor,

(B) $3 \leq |S| \leq h$,

(C) $G - S$ is connected, and

(D) $S = N_G(V(G) \setminus S)$.

Then there is a (rooted) tree decomposition (T, β) of G , a function $\gamma: V(T) \rightarrow 2^{V(G)}$, and a vertex-coloring λ such that

(I) $|V(T)| \leq 2 \cdot |V(G)|$,

(II) *the adhesion width of (T, β) is at most $h - 1$,*

(III) *for every $t \in V(T)$ with children t_1, \dots, t_ℓ , one of the following options holds:*

a. $\beta(t) \cap \beta(t_i) \neq \beta(t) \cap \beta(t_j)$ for all distinct $i, j \in [\ell]$, or

b. $\beta(t) = \beta(t) \cap \beta(t_i)$ for all $i \in [\ell]$,

(IV) $S \subsetneq \gamma(r)$ where r denotes the root of T ,

(V) $|\gamma(t)| \leq h^4$ for every $t \in V(T)$,

(VI) $\beta(t) \cap \beta(s) \subseteq \gamma(t) \subseteq \beta(t)$ for all $t \in V(T) \setminus \{r\}$, where s denotes the parent of t , and

(VII) $\beta(t) \subseteq \text{cl}_{2,2}^{(G,\lambda)}(\gamma(t))$ for all $t \in V(T)$.

Moreover, the decomposition (T, β) , the function γ , and the coloring λ can be computed in polynomial time, and the output is isomorphism-invariant with respect to (G, S, h) .

Proof. We give an inductive construction for the tree decomposition (T, β) as well as the function γ and the coloring λ . We start by arguing how to compute the set $\gamma(r)$.

► **Claim 12.** Let $v_1, v_2, v_3 \in S$ be three distinct vertices, and define $\chi := \chi_{\text{WL}}^1[G, S, v_1, v_2, v_3]$. Then there exists some $v \in V(G) \setminus S$ such that $|[v]_\chi| < h$.

Proof. Let $H := (G - (S \setminus \{v_1, v_2, v_3\})) - E(\{v_1, v_2, v_3\}, \{v_1, v_2, v_3\})$. It is easy to see that $\chi|_{V(H)}$ is 1-stable for the graph H . Observe that $H - \{v_1, v_2, v_3\} = G - S$ is connected. Suppose there is no vertex $v \in V(G) \setminus S$ such that $|[v]_\chi| < h$. Then χ is $(h-1)$ -CR-stable which implies that $\text{cl}_{h-1,1}^G(v_1, v_2, v_3) = \{v_1, v_2, v_3\}$. On the other hand, $Z := V(H) \setminus \{v_1, v_2, v_3\}$ induces a connected component of $H - \{v_1, v_2, v_3\}$, and $N_H(Z) = \{v_1, v_2, v_3\}$ since $S = N_G(V(G) \setminus S)$. But this contradicts Lemma 10. \blacktriangleleft

Let $v_1, v_2, v_3 \in S$ be distinct. We define $\chi[v_1, v_2, v_3] := \chi_{\text{WL}}^1[G, S, v_1, v_2, v_3]$. Also, let $c[v_1, v_2, v_3]$ denote the unique color such that

1. $c[v_1, v_2, v_3] \notin \{\chi[v_1, v_2, v_3](v) \mid v \in S\}$, and
2. $|(\chi[v_1, v_2, v_3])^{-1}(c[v_1, v_2, v_3])| \leq h - 1$

and which is minimal with respect to the linear order on the colors in the image of $\chi[v_1, v_2, v_3]$. Let $\gamma(v_1, v_2, v_3) := (\chi[v_1, v_2, v_3])^{-1}(c[v_1, v_2, v_3])$. Observe that $\gamma(v_1, v_2, v_3)$ is defined in an isomorphism-invariant manner given (G, S, h, v_1, v_2, v_3) . Now, define

$$\gamma(r) := S \cup \bigcup_{v_1, v_2, v_3 \in S \text{ distinct}} \gamma(v_1, v_2, v_3).$$

Clearly, $\gamma(r)$ is defined in an isomorphism-invariant manner given (G, S, h) . Moreover,

$$|\gamma(r)| \leq |S| + |S|^3 \cdot (h - 1) \leq |S|^3 \cdot h \leq h^4.$$

Finally, define $\beta(r) := \text{cl}_{2,2}^G(\gamma(r))$.

Let Z_1, \dots, Z_ℓ be the connected components of $G - \beta(r)$. Also, let $S_i := N_G(Z_i)$ and G_i be the graph obtained from $G[S_i \cup Z_i]$ by turning S_i into a clique, $i \in [\ell]$. We have $|S_i| < h$ by Lemma 8. Also, $|S_i| \geq 3$ and G_i is 3-connected since G is 3-connected. Clearly, $G_i - S_i$ is connected and $S_i = N_{G_i}(V(G_i) \setminus S_i)$. Finally, $G_i - E(S_i, S_i)$ excludes $K_{3,h}$ as a minor because $G - E(S, S)$ excludes $K_{3,h}$ as a minor.

We wish to apply the induction hypothesis to the triples (G_i, S_i, h) . If $|V(G_i)| = |V(G)|$ then $\ell = 1$ and $S \subsetneq S_i$. In this case the algorithm still makes progress since the size of S can be increased at most $h - 3$ times.

By the induction hypothesis, there are tree decompositions (T_i, β_i) of G_i and functions $\gamma_i: V(T_i) \rightarrow 2^{V(G_i)}$ satisfying Properties I - VII. We define (T, β) to be the tree decomposition where T is obtained from the disjoint union of T_1, \dots, T_ℓ by adding a fresh root vertex r which is connected to the root vertices of T_1, \dots, T_ℓ . Also, $\beta(r)$ is defined as above and $\beta(t) := \beta_i(t)$ for all $t \in V(T_i)$ and $i \in [\ell]$. Finally, $\gamma(r)$ is again defined as above, and $\gamma(t) := \gamma_i(t)$ for all $t \in V(T_i)$ and $i \in [\ell]$.

The algorithm clearly runs in polynomial time and the output is isomorphism-invariant (the coloring λ is defined below). We need to verify that Properties I - VII are satisfied. Using the comments above and the induction hypothesis, it is easy to verify that Properties II, IV, V and VI are satisfied.

For Property VII it suffices to ensure that $\text{cl}_{2,2}^{(G_i, \lambda)}(\gamma(t)) \subseteq \text{cl}_{2,2}^{(G, \lambda)}(\gamma(t))$. Towards this end, it suffices to ensure that $\lambda(v) \neq \lambda(w)$ for all $v \in \beta(r)$ and $w \in V(G) \setminus \beta(r)$. To ensure this property holds on all levels of the tree, we can simply define $\lambda(v) := \{\text{dist}_T(r, t) \mid t \in V(T), v \in \beta(t)\}$.

Next, we modify the tree decomposition in order to ensure Property III. Consider a node $t \in V(T)$ with children t_1, \dots, t_ℓ . We say that $t_i \sim t_j$ if $\beta(t) \cap \beta(t_i) = \beta(t) \cap \beta(t_j)$. Let A_1, \dots, A_k be the equivalence classes of the equivalence relation \sim . For every $i \in [k]$ we introduce a fresh node s_i . Now, every $t_j \in A_i$ becomes a child of s_i and s_i becomes a child of t . Finally, we set $\beta(s_i) = \gamma(s_i) := \beta(t) \cap \beta(t_j)$ for some $t_j \in A_i$. Observe that after this modification, Properties II and IV - VII still hold.

Finally, it remains to verify Property I. Before the modification described in the last paragraph, we have that $|V(T)| \leq |V(G)|$. Since the modification process at most doubles the number of nodes in T , the bound follows. \blacktriangleleft

7 An FPT Isomorphism Test for Graphs of Small Genus

Building on the decomposition theorem given in the last section, we can now prove the main result of this paper.

► **Theorem 13.** *Let G_1, G_2 be two (vertex- and arc-colored) graphs that exclude $K_{3,h}$ as a minor. Then one can decide whether G_1 is isomorphic to G_2 in time $2^{\mathcal{O}(h^4 \log h)} n^{\mathcal{O}(1)}$.*

Proof Idea. Suppose $G_i = (V(G_i), E(G_i), \chi_V^i, \chi_E^i)$ for $i \in \{1, 2\}$. Using standard reduction techniques (see, e.g., [11]) we may assume without loss generality that G_1 and G_2 are 3-connected. Pick an arbitrary set $S_1 \subseteq V(G_1)$ such that $|S_1| = 3$ and $G_1 - S_1$ is connected. For every $S_2 \subseteq V(G_2)$ such that $|S_2| = 3$ and $G_2 - S_2$ is connected, the algorithm tests whether there is an isomorphism $\varphi: G_1 \cong G_2$ such that $S_1^\varphi = S_2$. Observe that $S_i = N_{G_i}(V(G_i) \setminus S_i)$ for both $i \in \{1, 2\}$ since G_1 and G_2 are 3-connected. This implies that the triple (G_i, S_i, h) satisfies the requirements of Theorem 11. Let (T_i, β_i) be the tree decomposition, $\gamma_i: V(T_i) \rightarrow 2^{V(G_i)}$ be the function, and λ_i be the vertex-coloring computed by Theorem 11 on input (G_i, S_i, h) .

Now, the basic idea is compute isomorphisms between (G_1, S_1) and (G_2, S_2) using dynamic programming along the tree decompositions. More precisely, we aim at recursively computing the set

$$\Lambda := \text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2))[S_1]$$

(here, $\text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2))$ denotes the set of isomorphisms $\varphi: G_1 \cong G_2$ which additionally respect the vertex-colorings λ_i and satisfy $S_1^\varphi = S_2$). Throughout the recursive algorithm, we maintain the property that $|S_i| \leq h$. Also, we may assume without loss of generality that S_i is λ_i -invariant (otherwise, we replace λ_i by λ'_i defined via $\lambda'_i(v) := (1, \lambda_i(v))$ for all $v \in S_i$, and $\lambda'_i(v) := (0, \lambda_i(v))$ for all $v \in V(G_i) \setminus S_i$).

Let r_i denote the root node of T_i . Let ℓ denote the number of children of r_i in the tree T_i (if the number of children is not the same, the algorithm concludes that $\text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2)) = \emptyset$). Let t_1^i, \dots, t_ℓ^i be the children of r_i in T_i , $i \in \{1, 2\}$. For $i \in \{1, 2\}$ and $j \in [\ell]$ let V_j^i denote the set of vertices appearing in bags below (and including) t_j^i . Also let $S_j^i := \beta_i(r_i) \cap \beta_i(t_j^i)$ be the adhesion set to the j -th child, and define $G_j^i := G_i[V_j^i]$. Finally, let T_j^i denote the subtree of T_i rooted at node t_j^i , and $\beta_j^i := \beta_i|_{V(T_j^i)}$, $\gamma_j^i := \gamma_i|_{V(T_j^i)}$ and $\lambda_j^i := \lambda_i|_{V_j^i}$.

For every $i, i' \in \{1, 2\}$, and every $j, j' \in [\ell]$, the algorithm recursively computes the set

$$\Lambda_{j,j'}^{i,i'} := \text{Iso}((G_j^i, \lambda_j^i, S_j^i), (G_{j'}^{i'}, \lambda_{j'}^{i'}, S_{j'}^{i'}))[S_j^i].$$

We argue how to compute the set Λ . Building on Theorem 11, Item III, we may assume that

(a) $S_j^i \neq S_{j'}^i$ for all distinct $j, j' \in [\ell]$ and $i \in \{1, 2\}$, or

(b) $\beta(r_i) = S_j^i$ for all $j \in [\ell]$ and $i \in \{1, 2\}$

(if r_1 and r_2 do not satisfy the same option, then $\text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2)) = \emptyset$).

We first cover Option b. In this case $|\beta(r_i)| = |S_j^i| \leq h - 1$ by Theorem 11, Item II. The algorithm iterates over all bijections $\sigma: \beta(r_1) \rightarrow \beta(r_2)$. Now,

$$\sigma \in \text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2))[\beta(r_1)] \Leftrightarrow \exists \rho \in \text{Sym}([\ell]) \forall j \in [\ell]: \sigma \in \Lambda_{j, \rho(j)}^{1,2}.$$

To test whether σ satisfies the right-hand side condition, the algorithm constructs an auxiliary graph H_σ with vertex set $V(H_\sigma) := \{1, 2\} \times [\ell]$ and edge set

$$E(H_\sigma) := \{(1, j)(2, j') \mid \sigma \in \Lambda_{j,j'}^{1,2}\}.$$

Observe that H_σ is bipartite with bipartition $(\{1\} \times [\ell], \{2\} \times [\ell])$. Now,

$$\sigma \in \text{Iso}((G_1, \lambda_1, S_1), (G_2, \lambda_1, S_2))[\beta(r_1)] \Leftrightarrow H_\sigma \text{ has a perfect matching.}$$

It is well-known that the latter can be checked in polynomial time. This completes the description of the algorithm in case Option b is satisfied.

Next, suppose Option a is satisfied. Here, the central idea is to construct auxiliary vertex- and arc-colored graphs $H_i = (V(H_i), E(H_i), \mu_V^i, \mu_E^i)$ and sets $A_i \subseteq V(H_i)$ such that

1. $\beta_i(r_i) \subseteq A_i$ and $A_i \subseteq \text{cl}_{2,2}^{H_i}(\gamma_i(r_i))$, and
2. $\text{Iso}(H_1, H_2)[S_1] = \text{Iso}(H_1[A_1], H_2[A_2])[S_1] = \Lambda$.

Towards this end, we set

$$V(H_i) := V(G_i) \uplus \{(S_j^i, \gamma) \mid j \in [\ell], \gamma \in \Lambda_{j,j}^{i,i}\}$$

and

$$E(H_i) := E(G_i) \cup \{(S_j^i, \gamma)v \mid j \in [\ell], \gamma \in \Lambda_{j,j}^{i,i}, v \in S_j^i\}.$$

Also, we set

$$A_i := \beta(r_i) \cup \{(S_j^i, \gamma) \mid j \in [\ell], \gamma \in \Lambda_{j,j}^{i,i}\}.$$

The main idea is to use the additional vertices attached to the set S_j^i to encode the isomorphism type of the graph $(G_j^i, \lambda_j^i, S_j^i)$. This information is encoded by the vertex- and arc-coloring building on sets $\Lambda_{j,j}^{i,i'}$ already computed above. Let $\mathcal{S} := \{S_j^i \mid i \in \{1, 2\}, j \in [\ell]\}$, and define $S_j^i \sim S_{j'}^{i'}$ if $\Lambda_{j,j'}^{i,i'} \neq \emptyset$. Observe that \sim is an equivalence relation. Let $\{\mathcal{P}_1, \dots, \mathcal{P}_k\}$ be the partition of \mathcal{S} into the equivalence classes. We set

$$\mu_V^i(v) := (0, \chi_V^i(v), \lambda_i(v))$$

for all $v \in S_i$,

$$\mu_V^i(v) := (1, \chi_V^i(v), \lambda_i(v))$$

for all $v \in \gamma_i(r_i) \setminus S_i$,

$$\mu_V^i(v) := (2, \chi_V^i(v), \lambda_i(v))$$

for all $v \in \beta_i(r_i) \setminus \gamma_i(r_i)$,

$$\mu_V^i(v) := (3, \chi_V^i(v), \lambda_i(v))$$

for all $v \in V(G_i) \setminus \beta_i(r_i)$, and

$$\mu_V^i(S_j^i, \gamma) := (4, q, q)$$

for all $q \in [k]$, $S_j^i \in \mathcal{P}_q$, and $\gamma \in \Lambda_{j,j}^{i,i}$. For every $q \in [k]$ fix some $i(q) \in \{1, 2\}$ and $j(q) \in [\ell]$ such that $S_{j(q)}^{i(q)} \in \mathcal{P}_q$ (i.e., for each equivalence class, the algorithm fixes one representative). Also, for every $q \in [k]$ and $S_j^i \in \mathcal{P}_q$, fix a bijection $\sigma_j^i \in \Lambda_{j(q),j}^{i(q),i}$ such that $\sigma_{j(q)}^{i(q)}$ is the identity mapping. Finally, for $q \in [k]$, fix a numbering $S_{j(q)}^{i(q)} = \{u_1^q, \dots, u_{s(q)}^q\}$.

With this, we are ready to define the arc-coloring μ_E^i . First, we set

$$\mu_E^i(v, w) := (0, \chi_E^i(v, w))$$

for all $vw \in E(G_i)$. Next, consider an edge $(S_j^i, \gamma)v$ where $j \in [\ell]$, $\gamma \in \Lambda_{j,j}^{i,i}$, and $v \in S_j^i$. Suppose $S_j^i \in \mathcal{P}_q$. We set

$$\mu_E^i(v, (S_j^i, \gamma)) = \mu_E^i((S_j^i, \gamma), v) := (1, c)$$

for the unique $c \in [s(q)]$ such that

$$v = (u_c^q)^{\sigma_j^i \gamma}.$$

This completes the description of the graphs H_i and the sets A_i , $i \in \{1, 2\}$. It can be checked that Properties 1 and 2 are satisfied (see the full version for details).

Now, recall that the algorithm aims at computing the set Λ . Building on Property 2, we can simply compute $\text{Iso}(H_1[A_1], H_2[A_2])[S_1]$. Towards this end, the algorithm iterates through all bijections $\tau: \gamma_1(r_1) \rightarrow \gamma_2(r_2)$, and wishes to test whether there is an isomorphism $\varphi \in \text{Iso}(H_1[A_1], H_2[A_2])$ such that $\varphi[\gamma_1(r_1)] = \tau$. Note that, since $\gamma_i(r_i)$ is μ_V^i -invariant, it now suffices to solve this latter problem.

So fix a bijection $\tau: \gamma_1(r_1) \rightarrow \gamma_2(r_2)$ (if $|\gamma_1(r_1)| \neq |\gamma_2(r_2)|$ then the algorithm returns $\Lambda = \emptyset$). Let $\mu_1^*(v) := (1, v)$ for all $v \in \gamma_1(r_1)$, $\mu_1^*(v) := (0, \mu_V^1)$ for all $v \in V(H_1) \setminus \gamma_1(r_1)$, and $\mu_2^*(v) := (1, \tau^{-1}(v))$ for all $v \in \gamma_2(r_2)$ and $\mu_2^*(v) := (0, \mu_V^2)$ for all $v \in V(H_2) \setminus \gamma_2(r_2)$.

Intuitively speaking, μ_1^* and μ_2^* are obtained from μ_V^1 and μ_V^2 by individualizing all vertices from $\gamma_1(r_1)$ and $\gamma_2(r_2)$ according to the bijection τ . Now, we can apply Theorem 6 on input graph $H_1^* = (H_1, \mu_1^*)$ and $H_2^* = (H_2, \mu_2^*)$, and parameters $t = k := 2$.

Building on Property 1, we obtain a $\widehat{\Gamma}_2$ -group $\Gamma \leq \text{Sym}(A_1)$ and a bijection $\theta: A_1 \rightarrow A_2$ such that $\text{Iso}(H_1^*[A_1], H_2^*[A_2]) \subseteq \Gamma\theta$. Now, we can determine whether $H_1^*[A_1] \cong H_2^*[A_2]$ using Theorem 3. Using Property 2, this provides the answer to whether $\tau[S_1] \in \Lambda$ (recall that $S_1 \subseteq \gamma_1(r_1)$ by Theorem 11, Items IV and VI).

Overall, this completes the description of the algorithm. It only remains to analyse its running time. Let n denote the number of vertices of G_1 and G_2 .

The algorithm iterates over at most n^3 choices for the initial set S_2 , and computes the decompositions (T_i, β_i) , the functions γ_i , and the colorings λ_i in polynomial time. For the dynamic programming tables, the algorithm needs to compute $\mathcal{O}(n^2)$ many Λ -sets (using Theorem 11, Item I), each of which contains at most $h! = 2^{\mathcal{O}(h \log h)}$ many elements by Theorem 11, Item II. Hence, it remains to analyse the time required to compute the set Λ given the $\Lambda_{j,j'}^{i,i'}$ -sets. For Option b, this can clearly be done in time $2^{\mathcal{O}(h \log h)} n^{\mathcal{O}(1)}$.

So consider Option a. The graph H_i can clearly be computed in time polynomial in its size. We have that $|V(H_i)| = 2^{\mathcal{O}(h \log h)} n$. Afterwards, the algorithm iterates over $|\gamma_1(r_1)|!$ many bijections τ . By Theorem 11, Item V, we have that $|\gamma_1(r_1)|! = 2^{\mathcal{O}(h^4 \log h)}$. For each bijection, the algorithm then requires polynomial computation time by Theorems 6 and 3. Overall, this proves the bound on the running time. \blacktriangleleft

► **Remark 14.** The algorithm from the last theorem can be extended in two directions. First, if one of the input graphs does not exclude $K_{3,h}$ as a minor, it can be modified to either correctly conclude that G_1 has a minor isomorphic to $K_{3,h}$, or to correctly decide whether G_1 is isomorphic to G_2 . Indeed, the only part of the algorithm that exploits that the input graphs do not have a minor isomorphic to $K_{3,h}$ is the computation of the tree decompositions (T_i, β_i) from Theorem 11. In turn, this theorem only exploits forbidden minors via Lemmas 8 and 10. An algorithm can easily detect if one of the implications of those two statements is violated, in which case it can infer the existence of a minor $K_{3,h}$.

Secondly, using standard reduction techniques (see, e.g., [19]), one can also compute a representation of the set of all isomorphisms $\text{Iso}(G_1, G_2)$ in the same time.

Since every graph of Euler genus g excludes $K_{3,4g+3}$ as a minor [25], we obtain the following corollary.

► **Corollary 15.** *Let G_1, G_2 be two (vertex- and arc-colored) graphs of Euler genus at most g . Then one can decide whether G_1 is isomorphic to G_2 in time $2^{\mathcal{O}(g^4 \log g)} n^{\mathcal{O}(1)}$.*

8 Conclusion

We presented an isomorphism test for graphs excluding $K_{3,h}$ as a minor running in time $2^{\mathcal{O}(h^4 \log h)} n^{\mathcal{O}(1)}$. For this, we provided a polynomial-time isomorphism algorithm for (t, k) -WL-bounded graphs and argued that graphs excluding $K_{3,h}$ as a minor can be decomposed into parts that are $(2, 2)$ -WL-bounded after individualizing a small number of vertices.

Still, several questions remain open. Probably one of the most important questions in the area is whether isomorphism testing for graphs excluding K_h as a minor is fixed-parameter tractable with parameter h . As graphs of bounded genus form an important subclass of graphs excluding K_h as a minor, the techniques developed in this paper might also prove helpful in resolving this question.

As an intermediate step, one can also ask for an isomorphism test for graphs excluding $K_{\ell,h}$ as a minor running in time $f(h, \ell) n^{g(\ell)}$ for some functions f, g . Observe that this paper provides such an algorithm for $\ell = 3$. Indeed, combining ideas from [10, 22] with the approach taken in this paper, it seems the only hurdle towards such an algorithm is a generalization of Lemma 9. Given a connected graph G for which $|V_c| \geq \ell$ for all $c \in C_V(G, \chi_{\text{WL}}^2[G])$, is it always possible to find vertex-disjoint, connected subgraphs $H_1, \dots, H_\ell \subseteq G$ such that $V(H_r) \cap V_c \neq \emptyset$ for all $r \in [\ell]$ and $c \in C_V(G, \chi_{\text{WL}}^2[G])$?

As another intermediate problem, one can also consider the class \mathcal{G}_h of all graphs G for which there is a set $X \subseteq V(G)$ of size $|X| \leq h$ such that $G - X$ is planar. Is isomorphism testing fixed-parameter tractable on \mathcal{G}_h parameterized by h ?

References

- 1 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
- 2 László Babai, Peter J. Cameron, and Péter P. Pálffy. On the orders of primitive groups with restricted nonabelian composition factors. *J. Algebra*, 79(1):161–168, 1982. doi:10.1016/0021-8693(82)90323-4.
- 3 László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 162–171. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.10.
- 4 Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990. doi:10.1016/0196-6774(90)90013-5.
- 5 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.
- 6 John D. Dixon and Brian Mortimer. *Permutation Groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996. doi:10.1007/978-1-4612-0731-3.
- 7 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015. doi:10.1137/120892234.
- 8 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An improved isomorphism test for bounded-tree-width graphs. *ACM Trans. Algorithms*, 16(3):34:1–34:31, 2020. doi:10.1145/3382082.
- 9 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1010–1029. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.66.

- 10 Martin Grohe, Daniel Wiebking, and Daniel Neuen. Isomorphism testing for graphs excluding small minors. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 625–636. IEEE, 2020. doi:10.1109/FOCS46700.2020.00064.
- 11 John E. Hopcroft and Robert Endre Tarjan. Isomorphism of planar graphs. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 131–152. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_13.
- 12 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.
- 13 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 14 Ken-ichi Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. *CoRR*, abs/1511.02460, 2015. arXiv:1511.02460.
- 15 Ken-ichi Kawarabayashi and Bojan Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 471–480. ACM, 2008. doi:10.1145/1374376.1374443.
- 16 Ken-ichi Kawarabayashi, Bojan Mohar, Roman Nedela, and Peter Zeman. Automorphism groups of maps in linear time. *CoRR*, abs/2008.01616, 2020. arXiv:2008.01616.
- 17 Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.*, 46(1):161–189, 2017. doi:10.1137/140999980.
- 18 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.
- 19 Rudolf Mathon. A note on the graph isomorphism counting problem. *Inf. Process. Lett.*, 8(3):131–132, 1979. doi:10.1016/0020-0190(79)90004-8.
- 20 Gary L. Miller. Isomorphism of graphs which are pairwise k-separable. *Inf. Control.*, 56(1/2):21–33, 1983. doi:10.1016/S0019-9958(83)80048-5.
- 21 Daniel Neuen. Hypergraph isomorphism for groups with restricted composition factors. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 88:1–88:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.88.
- 22 Daniel Neuen. Isomorphism testing for graphs excluding small topological subgraphs. *CoRR*, abs/2011.14730, 2020. arXiv:2011.14730.
- 23 Ilia N. Ponomarenko. The isomorphism problem for classes of graphs. *Dokl. Akad. Nauk SSSR*, 304(3):552–556, 1989.
- 24 Ilia N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, June 1991. doi:10.1007/BF01098279.
- 25 Gerhard Ringel. Das geschlecht des vollständigen paaren graphen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 28(3):139–150, October 1965. doi:10.1007/BF02993245.
- 26 Joseph J. Rotman. *An Introduction to the Theory of Groups*, volume 148 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1995. doi:10.1007/978-1-4612-4176-8.

- 27 Ákos Seress. *Permutation Group Algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003. doi:10.1017/CB09780511546549.
- 28 Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 1968. English translation by Grigory Ryabov available at https://www.itl.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Restricted t -Matchings via Half-Edges

Katarzyna Paluch

Institute of Computer Science, University of Wrocław, Poland

Mateusz Wasylikiewicz

Institute of Computer Science, University of Wrocław, Poland

Abstract

For a bipartite graph G we consider the problem of finding a maximum size/weight square-free 2-matching and its generalization - the problem of finding a maximum size/weight $K_{t,t}$ -free t -matching, where t is an integer greater than two and $K_{t,t}$ denotes a bipartite clique with t vertices on each of the two sides. Since the weighted versions of these problems are \mathcal{NP} -hard in general, we assume that the weights are vertex-induced on any subgraph isomorphic to $K_{t,t}$. We present simple combinatorial algorithms for these problems. Our algorithms are significantly simpler and faster than those previously known. We dispense with the need to shrink squares and, more generally subgraphs isomorphic to $K_{t,t}$, the operation which occurred in all previous algorithms for such t -matchings and instead use so-called *half-edges*. A *half-edge* of edge e is, informally speaking, a half of e containing exactly one of its endpoints.

Additionally, we consider another problem concerning restricted matchings. Given a (not necessarily bipartite) graph $G = (V, E)$, a set of k subsets of edges E_1, E_2, \dots, E_k and k natural numbers r_1, r_2, \dots, r_k , the Restricted Matching Problem asks to find a maximum size matching of G among such ones that for each $1 \leq i \leq k$, M contains at most r_i edges of E_i . This problem is \mathcal{NP} -hard even when G is bipartite. We show that it is solvable in polynomial time if (i) for each i the graph G contains a clique or a bipartite clique on all endpoints of E_i ; in the case of a bipartite clique it is required to contain E_i and (ii) the sets E_1, \dots, E_k are almost vertex-disjoint - the endpoints of any two different sets have at most one vertex in common.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases restricted 2-matchings

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.73

Funding Partially supported by Polish National Science Center grant 2018/29/B/ST6/02633.

Acknowledgements The authors thank Pratik Ghosal and Prajakta Nimbhorkar for discussions at the early stage of research on this paper.

1 Introduction

Given a positive integer t , a subset M of edges of an undirected simple graph G is called a t -matching if every vertex is incident to at most t edges of M . A t -matching of maximum size can be found in polynomial time by a reduction to the classical matching problem. A 2-matching is called *square-free* if it does not contain any cycle of length 4. A C_k -free 2-matching is one without any cycle of length at most k . The C_k -free 2-matching problem consists in finding a C_k -free 2-matching of maximum size. Observe that the C_k -free 2-matching problem for $n/2 \leq k < n$, where n is the number of vertices in the graph, is equivalent to finding a Hamiltonian cycle, and thus \mathcal{NP} -hard. Hartvigsen [13] gave a complicated algorithm for the case of $k = 3$. Papadimitriou [4] showed that this problem is \mathcal{NP} -hard when $k \geq 5$. The complexity of the C_4 -free 2-matching problem is unknown.

When the graph is bipartite the smallest length of a cycle contained in it is at least 4. We refer to cycles of length four as *squares*. Polynomial time algorithms for the C_4 -free 2-matching problem in bipartite graphs were shown by Hartvigsen [14], Pap [30], Babenko [1]



© Katarzyna Paluch and Mateusz Wasylikiewicz;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 73; pp. 73:1–73:17

Leibniz International Proceedings in Informatics



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

and analyzed by Király [18]. A generalization of a square-free 2-matching in a bipartite graph is a $K_{t,t}$ -free t -matching - a t -matching without any $K_{t,t}$ - a bipartite clique with t vertices on each of the two sides. $K_{t,t}$ -free t -matchings were first considered by Frank [7], who provided a min-max formula for $K_{t,t}$ -free t -matchings based on a result in [8] on crossing bi-supermodular functions. Using this formula, it is possible to compute the size of a maximum $K_{t,t}$ -free t -matching by the ellipsoid method or a combinatorial method by Fleiner [6]. Moreover, one can compute a maximum $K_{t,t}$ -free t -matching through Végh and Benczúr's algorithm [36] for covering pairs of sets and directly using Pap's algorithm [29].

In the weighted version of the $K_{t,t}$ -free t -matching problem, each edge e is associated with a nonnegative weight $w(e)$ and we are interested in finding a $K_{t,t}$ -free t -matching of maximum weight, where the weight of a t -matching M is defined as the sum of weights of edges belonging to M . The weighted square-free 2-matching problem in bipartite graphs was proven to be \mathcal{NP} -hard [12, 19] even if the weight of every edge is either 0 or 1. Bérczi and Kobayashi [2] sharpened the result and showed that the problem is \mathcal{NP} -hard even if the given graph is cubic, bipartite and planar. The weighted $K_{t,t}$ -free t -matching problem in bipartite graphs is solvable in polynomial time when the weights of edges are vertex-induced on every subgraph isomorphic to $K_{t,t}$, which was shown by Makai [24] and Takazawa [33].

Apart from $K_{t,t}$ -free t -matchings, we consider another problem concerning restricted matchings. Given a (not necessarily bipartite) graph $G = (V, E)$, a set of k subsets of edges E_1, E_2, \dots, E_k and k natural numbers r_1, r_2, \dots, r_k , the Restricted Matching Problem asks to find a maximum size classical matching of G among such ones that for each $1 \leq i \leq k$, M contains at most r_i edges of E_i . This problem was first studied in [17] by Itai, Rodeh and Tanimoto for bipartite graphs and shown to be \mathcal{NP} -hard for the general case and solvable in polynomial time for the variant when there is only one set E_1 , i.e., when $k = 1$. The version, in which G is bipartite and each E_i contains two edges (and hence each $r_i = 1$) was proven to be \mathcal{NP} -hard by Garey and Johnson [11]. The problem was also considered in [25] and [31].

Our results We present simple combinatorial algorithms for the weighted and unweighted version of the $K_{t,t}$ -free t -matching problem in bipartite graphs. In the weighted version we assume that the weights of edges are vertex-induced on every subgraph isomorphic to $K_{t,t}$. In these algorithms we successively find and apply a minimum length/weight augmenting path until it is no longer possible. The search for an augmenting path is conducted in a specially modified graph G , called G' . Graph G' is obtained from G by replacing some subgraphs with so-called gadgets that contain *half-edges*. A *half-edge* of edge e is, informally speaking, a half of e containing exactly one of its endpoints. Half-edges have been first introduced in [27]. Each subgraph that is replaced with a gadget in a given step is isomorphic to $K_{t,t}$ and $t^2 - 1$ of its edges belong to the current t -matching M . In previous algorithms for these problems such or similar subgraphs were *shrunk*. One could say that we take an opposite approach and *expand* such subgraphs. However, in our case these expansions do not build on each other and in each step G' is constructed only from the original graph G and a current t -matching M . We give a detailed description of these algorithms for square-free 2-matchings and their analyses and only an outline for the $K_{t,t}$ -free t -matching problem. In addition to being significantly simpler our algorithms are also faster than those known previously. For the unweighted square-free 2-matching problem our algorithm has running time $\mathcal{O}(nm)$, where n denotes the number of vertices in the graph and m the number of edges. Both algorithms by Hartvigsen and Babenko run in $\mathcal{O}(n^3)$ time and the one by Pap in $\mathcal{O}(n^4)$. For the weighted/unweighted version of the $K_{t,t}$ -free t -matching problem we give an algorithm with running time, respectively, $\mathcal{O}(tnm + t^2n^2 \log n)$ and $\mathcal{O}(nm + tn^2 + \sqrt{tnm})$. For the weighted variant the algorithm by Takazawa has runtime $\mathcal{O}(tn^2m + tn^3 \log n)$ for the t -factor

problem (each vertex has to be incident with exactly t edges) and $\mathcal{O}(t^5 n^6 + t^4 n^6 \log n)$ for the t -matching problem because of the costly reduction to the t -factor problem. The algorithm by Makai has polynomial time but it uses the ellipsoid method.

Regarding classical matchings we devise a polynomial time algorithm for the variant of the Restricted Matching Problem when (i) for each i the graph G contains a clique or a bipartite clique on all endpoints of E_i ; in the case of a bipartite clique it is required to contain E_i and (ii) the sets E_1, \dots, E_k are almost vertex-disjoint - the endpoints of any two different sets have at most one vertex in common.

Motivation C_k -free 2-matchings and $K_{t,t}$ -free t -matchings are classical problems of combinatorial optimization. They have applications in traveling salesman problems, problems related to finding a smallest 2-edge-connected spanning subgraph as well as in increasing the vertex-connectivity (see [5, 2, 3, 34] for more details). A good survey of these applications has been given by Takazawa [35].

Related work Some generalizations of the C_k -free 2-matching problem were investigated. Recently, Kobayashi [21] gave a polynomial algorithm for finding a maximum weight 2-matching that does not contain any triangle from a given set of forbidden edge-disjoint triangles. Polynomial algorithms for square-free and/or triangle-free 2-matchings in subcubic graphs were presented in [15, 16, 2, 3, 20, 28, 22]. An algorithm by Paluch and Wasylkiewicz uses a similar approach as the one presented in this paper but requires only one computation of a b -matching. When it comes to the square-free 2-matching problem in general graphs, Nam [26] constructed a complex algorithm for it for graphs, in which all squares are vertex-disjoint.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E . We denote the number of vertices of G by n and the number of edges of G by m . We denote a vertex set of G by $V(G)$ and an edge set by $E(G)$. We assume that all graphs are *simple*, i.e., they contain neither loops nor parallel edges. We denote an edge connecting vertices v and u by (v, u) . A *path* of graph G is a sequence $P = (v_0, \dots, v_l)$ for some $l \geq 1$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \{0, 1, \dots, l-1\}$. We refer to l as the *length* of P . A *cycle* of graph G is a sequence $c = (v_0, \dots, v_{l-1})$ for some $l \geq 3$ of pairwise distinct vertices of G such that $(v_i, v_{(i+1) \bmod l}) \in E$ for every $i \in \{0, 1, \dots, l-1\}$. We refer to l as the *length* of c . We will sometimes treat a path or a cycle as an edge set and sometimes as a sequence of edges. For an edge set $F \subseteq E$ and $v \in V$, we denote by $\deg_F(v)$ the number of edges of F incident to v . For any two edge sets $F_1, F_2 \subseteq E$, the symmetric difference $F_1 \oplus F_2$ denotes $(F_1 \setminus F_2) \cup (F_2 \setminus F_1)$.

For a natural number t , we say that an edge set $F \subseteq E$ is a *t -matching* if $\deg_F(v) \leq t$ for every $v \in V$. t -matchings belong to a wider class of b -matchings, where for every vertex v of G , we are given a natural number $b(v)$ and a subset of edges is a *b -matching* if every vertex v is incident to at most $b(v)$ of its edges. A b -matching of G of maximum weight can be computed in polynomial time. We refer to Lovász and Plummer [23] for further background on b -matchings.

Let M be a b -matching. We say that an edge e is *matched* (in M) if $e \in M$ and *unmatched* (in M) otherwise. Additionally, an edge belonging to M will be referred to as a *M -edge* and an edge not belonging to M as a *non- M -edge*. We call a vertex v *deficient (in M)* if $\deg_M(v) < b(v)$. An *M -alternating path* P is any sequence of vertices (v_1, v_2, \dots, v_k) such that edges on P are alternately M -edges and non- M -edges and no edge occurs on P

more than once and $v_1 \neq v_k$. An M -alternating cycle C has the same definition as an M -alternating path except that $v_1 = v_k$ and additionally $(v_{k-1}, v_k) \in M$ iff $(v_1, v_2) \notin M$. Note that an M -alternating path or cycle may go through some vertices more than once but via different edges. An M -alternating path is called M -augmenting if it begins and ends with a non- M -edge and if it begins and ends with a deficient vertex. We say that M is a **maximum** b -matching if there is no b -matching of G with more edges than M . Given a weight function $w : E \rightarrow \mathbb{R}$ we define **weight** of M as $w(M) = \sum_{e \in M} w(e)$. We say that M has **maximum weight** if there is no b -matching of G of weight greater than $w(M)$.

Given a weight function w , the **alternating weight** of an M -alternating path or cycle P is defined as $\tilde{w}(P) = \sum_{e \in P \cap M} w(e) - \sum_{e \in P \setminus M} w(e)$. We say that an M -augmenting path P is **minimum** if it has minimum alternating weight among all M -augmenting paths and cannot be shortened without increasing its alternating weight. An M -alternating cycle is a **negative cycle** if its alternating weight is negative. An **application** of an M -alternating path or cycle P to M is an operation whose result is $M \oplus P$. Note that $w(M \oplus P) = w(M) - \tilde{w}(P)$.

We are interested in computing a b -matching of a graph G where we are given vectors $l, u \in \mathbb{N}^V$ and a weight function $w : E \rightarrow \mathbb{R}$. For a vertex $v \in V$, $[l(v), u(v)]$ is said to be a **capacity interval** of v . An edge set $M \subseteq E$ is said to be an (l, u) -matching if $l(v) \leq \deg_M(v) \leq u(v)$ for every $v \in V$. A maximum weight (l, u) -matching can be computed efficiently.

For a weight function $w : E \rightarrow \mathbb{R}$ and a subgraph H of G , we say that w is **vertex-induced on H** if there exists a function $r : V(H) \rightarrow \mathbb{R}$ such that $w(u, v) = r(u) + r(v)$ for every edge (u, v) of H . We call r a **potential function** of H .

An instance of the square-free 2-matching problem consists of an undirected bipartite graph $G = (V, E)$ and the goal is to find a maximum square-free 2-matching of G . A generalization of a square-free 2-matching in a bipartite graph is a $K_{t,t}$ -free t -matching - a t -matching without any $K_{t,t}$ - a bipartite clique with t vertices on each of the two sides. An instance of the $K_{t,t}$ -free t -matching problem consists of an undirected bipartite graph $G = (V, E)$ and a natural number $t \geq 2$. The aim is to compute a maximum $K_{t,t}$ -free t -matching. We also consider weighted versions of these problems, in which we are additionally given a weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$ that is vertex-induced on each subgraph of G isomorphic to $K_{t,t}$ and the task consists in finding a maximum weight $K_{t,t}$ -free t -matching.

3 Outline of the Algorithm for Square-Free 2-Matchings

The general scheme of the algorithm for each variant of the square-free 2-matching problem is the same - we give it below.

■ **Algorithm 1** Computing a maximum (weight) square-free 2-matching of a bipartite graph G .

-
- 1: Let M be an empty 2-matching of G .
 - 2: Construct an auxiliary bipartite graph $G' = (V', E')$ with $\mathcal{O}(n)$ vertices and $\mathcal{O}(m)$ edges, and its 2-matching M' of size $\mathcal{O}(n)$ by replacing some squares of G with gadgets containing half-edges. (Both gadgets and half-edges are defined later.)
 - 3: Compute a shortest (resp. minimum) M' -augmenting path P of G' . If G' contains no M' -augmenting path (resp. no M' -augmenting path with negative alternating weight), stop the algorithm and return M .
 - 4: Apply P to M' obtaining M'' and extract a square-free 2-matching M_1 of G from M'' such that $|M_1| = |M| + 1$ (resp. $w(M_1) > w(M)$). Set M as M_1 and go to 2.
-

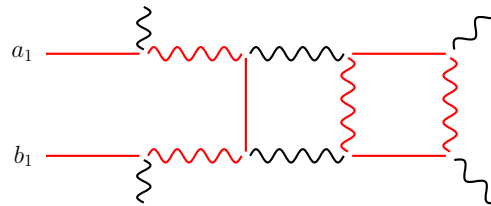
▷ **Claim 3.1.** Algorithm 1 runs in time $O(nm + n^2 \log n)$.

Proof. It will be easy to implement all steps of an Algorithm 1 except 3 in linear time. 3 can be implemented to run in time $O(|E'| + |V'| \log |V'|)$ by Fredman and Tarjan's implementation of Dijkstra's algorithm [9] similarly as in the Hungarian method described by Schrijver [32]. Every step is executed $O(n)$ times since $|M|$ increases by one every time 4 is executed. ◀

Let us also remark that in the unweighted version of the problem Algorithm 1 runs in $O(nm)$ since 3 can be implemented to run in linear time.

4 Maximum square-free 2-matchings in bipartite graphs

In this section we show how to find a maximum square-free 2-matching in a bipartite graph G . When computing a maximum 2-matching N of G , which is not required to be square-free, we can proceed as follows. As long as G contains some N -augmenting path P , apply it to N and repeat. When the goal is to compute a maximum square-free 2-matching of G , this approach is not applicable for two reasons. Firstly, by applying an augmenting path to a square-free 2-matching we may obtain a 2-matching which is not square-free. Secondly, it may happen that a square-free 2-matching M is not maximum but G does not contain any M -augmenting path P such that $M \oplus P$ is square-free. An example of such a 2-matching is shown in Figure 1. Nevertheless, it turns out as we demonstrate below, that we may still use this method if the search for an augmenting path is conducted in an appropriately modified graph G' .



■ **Figure 1** Edges of a square-free 2-matching M are waved. M is not maximum, G contains an M -augmenting path P (with endpoints a_1, b_1) but $M \oplus P$ is not square-free. If we apply both P and an M -alternating cycle C (indicated by red edges), we obtain a larger square-free 2-matching.

First let us check what types of squares of G are in danger of appearing in a 2-matching after the application of a shortest augmenting path. (In fact P does not have to be shortest - it suffices if P has no shortcuts.)

► **Fact 4.1.** *Any shortest M -augmenting path P' has the property that for any vertex v , it contains at most two edges incident to v : at most one matched edge and at most one unmatched one.*

Proof. Otherwise P' would contain an alternating cycle C and hence could be shortened. (This property does not hold in non-bipartite graphs.) ◀

► **Lemma 4.2** (Proposition 2.1. in [33]). *Let M be a square-free 2-matching of G and P any shortest M -augmenting path. If $M \oplus P$ contains a square s , then exactly three edges of s belong to M .*

Proof. Suppose that $M \oplus P$ contains a square $s = (a, b, c, d)$ such that at least two of its edges do not belong to M , one of which is (a, b) . Therefore $(a, b) \in P$ and by the fact above we get that both (b, c) and (a, d) belong to M . Hence, $(c, d) \notin M$. Thus, the edges $(a, b), (c, d)$ belong to P and are connected in P by an odd-length alternating M -path R . R 's endpoints are either a and d or b and c . Also, R begins and ends with an M -edge. This means that P can be shortened by replacing R either with (a, d) or (b, c) - a contradiction. \blacktriangleleft

Suppose that M is a (possibly empty) square-free 2-matching of G . We say that a square s of G is **saturated** (in M) if exactly three edges of s are contained in M . The graph G' in which we are going to search for an augmenting path is obtained from the original graph G by replacing a subset of saturated squares with specially constructed subgraphs called *gadgets*. More details are given below.

One can observe that, since G is bipartite, any edge e of G belongs to at most two different saturated squares.

► **Definition 4.3.** A saturated square s , which has exactly one common edge with some other saturated square is said to be **unproblematic**. Otherwise, s is said to be **problematic**.

Unproblematic squares can be easily got rid of by replacing some edges with other ones as explained in more detail in the proof of Lemma 4.8.

► **Fact 4.4.** If two problematic squares have a common edge, then they share exactly two edges, both of which are in M . Any problematic square is non-edge-disjoint with at most one other problematic square.

Proof. Let s_1 and s_2 be two problematic squares of G with a common edge. Since they are problematic, they have at least two common edges. Note that s_1 and s_2 cannot share more than two edges because otherwise they would share all four vertices. However, two different squares cannot share all vertices because G is simple and bipartite. Let e_1, e_2 be the common edges of s_1 and s_2 . They cannot be vertex-disjoint because then again s_1 and s_2 would have four common vertices. Hence, e_1 and e_2 have a common vertex v .

Next we argue that both e_1 and e_2 must belong to M . Suppose to the contrary that $e_1 \notin M$. This implies that the endpoint v' of e_2 different from v is incident to three edges of M - a contradiction.

To see that s_1 cannot share an edge with a problematic square $s_3 \neq s_2$, observe that in such a case s_3 would have to share with s_1 exactly one of the edges of $\{e_1, e_2\}$ and additionally some edge e_3 . The edge e_3 cannot belong to s_2 because it is incident to a vertex of s_1 not contained in s_2 . Hence, s_3 and s_2 share exactly one edge, which implies that s_2 is not problematic - a contradiction. \blacktriangleleft

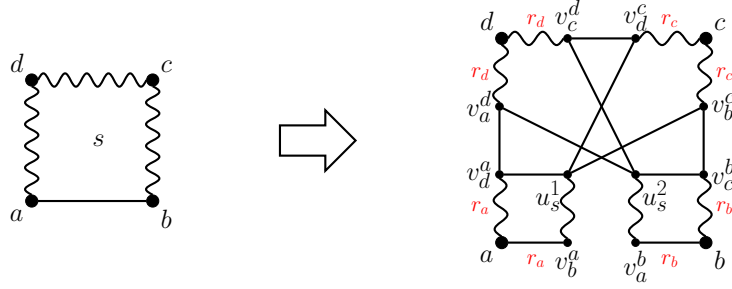
If G contains at least one problematic square, we build a graph $G' = (V', E')$ together with its 2-matching M' , in which each problematic square s is replaced with a subgraph, called a **gadget for** s . The precise construction of G' and M' is the following. We start off with $G' = G$. We initialize M' as the set of edges of M which are not part of any problematic square.

Let $s = (a, b, c, d)$ be any problematic square of G such that $(a, b) \notin M$. For each edge (p, q) of s we add two new vertices v_q^p and v_p^q , called **subdivision vertices** (of s), and we replace (p, q) with three new edges: $(p, v_q^p), (v_q^p, v_p^q), (v_p^q, q)$. Each of the edges $(p, v_q^p), (v_p^q, q)$ is called a **half-edge** (of (p, q) and also of s). The edge (v_q^p, v_p^q) is called an **eliminator** (of (p, q)). We remove the eliminator of the edge (a, b) from E' . Additionally, we introduce two new vertices u_s^1 and u_s^2 , called **global vertices**. We connect u_s^1 with every subdivision vertex

connected to a or c . Similarly, we connect u_s^2 with every subdivision vertex connected to b or d . We define vectors $l, u \in \mathbb{N}^{V'}$ as follows. We set a capacity interval of every vertex of the original graph G to $[0, 2]$ and we set a capacity interval of every other vertex of G' to $[1, 1]$, i.e., every vertex of $V' \setminus V$ is matched to exactly one vertex of G' in any (l, u) -matching of G' . For every edge $e \in M$ of s , we add half-edges of e to M' . Additionally, we add (u_s^1, v_b^a) and (u_s^2, v_a^b) to M' . If two problematic squares share two edges, then their gadgets overlap, i.e., we build a gadget for each one of them in the way described above.

The main ideas behind the gadget for a problematic square $s = (a, b, c, d)$ are the following. An (l, u) -matching M' of G' is to represent roughly a square-free 2-matching M of G . If M' contains both half-edges of some edge e , then e is included in M . If M' contains an eliminator of e , then e does not belong to M (is excluded from M). We want to ensure that at least one edge of s does not belong to M . This is done by requiring that the two global vertices u_s^1 and u_s^2 are matched to two subdivision vertices. In this way two half-edges of s are guaranteed not to belong to M' and hence to M .

Additionally, we can observe that for a 2-matching M depicted in Figure 1, there exists one M' -augmenting path in G' comprising all red edges.



■ **Figure 2** A gadget for a problematic square $s = (a, b, c, d)$ such that $(a, b) \notin M$. Weights of the edges for the weighted version are given in red.

It turns out that if G' contains an M' -augmenting path, then we can apply a shortest one to M' , obtaining a larger 2-matching M'' of G' . From M'' we can in turn obtain a square-free 2-matching M_2 of G of size $|M| + 1$. This is achieved by first changing around the half-edges of M'' so that for each edge $e \in E$ belonging to a problematic square we have that either both half-edges of e are contained in M'' or none. Next, if needed, we get rid of unproblematic squares.

► **Lemma 4.5.** *Let P be a shortest M' -augmenting path of G' and let $M'' = M' \oplus P$. Then there exists exactly one 2-matching M_1 of G , denoted as $\text{img}(M'')$, such that:*

1. *for each vertex v of G it holds that $\deg_{M''}(v) = \deg_{M_1}(v)$,*
2. *for each edge $e \in E$ not belonging to any problematic square, we have that $e \in M_1 \Leftrightarrow e \in M''$.*

Proof. We obtain M_1 from M'' as follows. First for each edge $e \in E$ not belonging to any problematic square, we include e into M_1 if and only if $e \in M''$. Next we remove all edges of M'' incident to global vertices and flip the half-edges so that for each edge $e \in E$ belonging to a problematic square we have that either both half-edges of e are contained in M'' or none. To see that the half-edges can indeed be changed around in such a way we use the following observation.

► **Observation 4.6.** *Let $s = (a, b, c, d)$ be a problematic square such that $(a, b) \notin M$. If P goes through any unmatched half-edge of (a, b) , then it does not go through any matched half-edge of s incident to a or b .*

Proof. Assume that P goes through (v_a^b, b) . Then P must also go through the global vertex u_s^2 , which in turn means that P must also go through d . (Otherwise P would contain an alternating cycle going through vertices u_s^2, v_c^b, b, v_a^b .) Suppose now that P also contains the matched half-edge of s incident to b . Then P must also contain (v_c^b, v_b^c) and (v_b^c, c) . (If s shares edges (b, c) and (c, d) with another problematic square s' , then from v_c^b P may go to a global vertex of s' , but then it also has to go to d and the case is as below.) This way we obtain a contradiction, because we could shorten P : instead of going through d and u_s^2, b, c using seven edges P could use three edges $(d, v_c^d), (v_c^d, v_d^c), (v_d^c, c)$ instead.

If, on the other hand, P also contained the matched half-edge incident to a , then it would have to contain also the eliminator of (a, d) and the second half-edge of (a, d) , which would mean that P contains four edges incident to d , using both matched edges incident to it - a contradiction. ◀

This means the following.

1. If P goes through u_s^1 but not u_s^2 , then P also goes through c and thus P contains exactly two half-edges of s : one matched half-edge incident to c and one unmatched half-edge incident to a . As a result M'' contains exactly one matched half-edge incident to c and two matched half-edges incident to a and thus the degrees of vertices a, b, c, d with respect to half-edges of s contained in M'' are equal to, respectively, 2, 1, 1, 2. In this case we set M_1 so that it contains edges $(b, a), (a, d), (d, c)$ but not (b, c) . The case when P goes through u_s^2 but not u_s^1 is symmetrical.
2. If P goes through both u_s^1 and u_s^2 , then P contains exactly four half-edges of s : one matched half-edge incident to c , one matched half-edge incident to d and both half-edges of (a, b) . As a result the degrees of vertices a, b, c, d with respect to half-edges of s contained in M'' are equal to, respectively, 2, 2, 1, 1. In this case we set M_1 to contain edges $(b, a), (a, d), (b, c)$ but not (d, c) .
3. If P goes neither through u_s^1 nor through u_s^2 , then P either does not go through any half-edge of s at all or goes through two matched half-edges of exactly one of the edges $(b, c), (c, d), (a, d)$. As a result M'' has the property that for each edge e of s either both half-edges of e are in M'' or none. In this case we include an edge e of s into M only if its both half-edges are present in M'' .

This finishes the proof. ◀

We have the analogue of Lemma 4.2.

► **Lemma 4.7.** *Let P be any shortest M' -augmenting path in G' . If $M_1 = \text{img}(M' \oplus P)$ contains a square s , then s is unproblematic (in M).*

Proof. Let $s = (a, b, c, d)$ be some square of G that is contained in M_1 . First, we can notice that if each of the unmatched edges of s is also present in G' , then by Lemma 4.2, we know that s can appear in M_1 only if it is saturated and hence only if it is unproblematic (because otherwise s is replaced with a gadget). Second, we observe that s cannot be problematic, because the gadget for s ensures that at least two half-edges of s do not belong to $M' \oplus P$ and hence at least one edge of s does not belong to M_1 .

Suppose then now that s is not saturated and at least one of its unmatched edges, say (a, b) , is not present in G' . It means that there exists a problematic square s' that contains (a, b) . The edge (a, b) appears in M_1 only if at least one of the half-edges of (a, b) is contained

in P . Suppose it is (a, v_b^a) . It means that P contains also some matched (half-)edge e' incident to a . By Observation 4.6 the edge e' cannot be contained in s' . Neither can it be contained in s because then s could not appear in M_1 . This means that the edge (a, d) of s is unmatched. We notice however, that (a, d) cannot belong to M_1 because neither (a, d) can belong to P (as it would mean that P contains four (half-)edges incident to a and thus could be shortened) nor any half-edge of (a, d) can belong to P (if (a, d) belongs to a problematic square s'' then by Observation 4.6, if P contains a half-edge of (a, d) , then it does not contain a matched half-edge incident to a). ◀

► **Lemma 4.8.** *Let P be any shortest M' -augmenting path in G' and $M_1 = \text{img}(M' \oplus P)$. If M_1 is not square-free then it can be transformed into a square-free 2-matching M_2 such that $|M_1| = |M_2|$.*

Proof. We consider every square $s = (a, b, c, d)$ of M_1 . By Lemma 4.7, s is unproblematic. Hence it shares exactly one edge with another unproblematic square s' . Assume that (a, b) is unmatched in M . Observe that (c, d) cannot be a common edge of s and s' because any vertex of G can be incident to at most two edges of M . Neither can (a, b) be a common edge of s and s' , because then P could be shortened and not go through (a, b) at all. Suppose then that (b, c) is a common edge of s and $s' = (b, c, e, f)$. It means that $(c, e) \notin M$ and $(b, f) \in P \cap M$. Since apart from (a, b) none of the edges of s belongs to P , the edge (e, c) cannot belong to P either. Therefore, s' is an M_1 -alternating cycle. We apply s' to M_1 . As a result s does not occur in M_1 any more. Also, this operation does not introduce any new square into M_1 because the edges $(f, b), (b, a), (a, d), (d, c), (c, e)$ form a path of length five and are guaranteed to belong to M_1 ; therefore, none of them can be part of a square. ◀

► **Lemma 4.9.** *If there is no M' -augmenting path in G' , then M is a maximum square-free 2-matching of G .*

Proof. It is a special case of Lemma 5.9. ◀

5 Maximum weight square-free 2-matchings in bipartite graphs

In this section we extend the results from the previous section to the weighted setting. Recall that this problem is \mathcal{NP} -hard for general weights, therefore we assume that the weight function w is vertex-induced on every square. Some proofs are omitted due to space constraints.

► **Lemma 5.1.** *Let s and s' be two problematic squares that have exactly two common edges. Then w is vertex-induced on $s \cup s'$.*

Proof. Let $s = (a, b, c, d)$ and $s' = (a, b, c, e)$. Let r and r' be potential functions of, respectively, s and s' . Observe that $r(a) + r(b) = w(a, b) = r'(a) + r'(b)$ and $r(b) + r(c) = w(b, c) = r'(b) + r'(c)$. Let $\Delta = r(a) - r'(a)$. We increase both $r'(a)$ and $r'(c)$ by Δ and decrease both $r'(b)$ and $r'(e)$ by Δ . Notice that r' is still a valid potential function of s' after this operation. Additionally, now r and r' agree on the common vertices. Therefore, $r \cup r'$ is a potential function of $s \cup s'$. ◀

To the construction of G' from Section 4 we add a weight function $w' : E(G') \rightarrow \mathbb{R}$ defined as follows. The half-edges incident to a, b, c and d get weight $r(a), r(b), r(c)$ and $r(d)$, respectively, where $r(a), \dots, r(d)$ are potentials of s . All the other edges of the gadget get weight 0.

73:10 Restricted t -Matchings via Half-Edges

Define $k : E(G') \rightarrow \{0, 1/2, 1\}$ such that

$$k(e) = \begin{cases} 1 & \text{if } e \in E(G), \\ 1/2 & \text{if } e \text{ is a half-edge,} \\ 0 & \text{otherwise.} \end{cases}$$

For $e \in E(G')$ we say that $k(e)$ is the **size** of e .

► **Definition 5.2.** Consider any (l, u) -matching N of G' . We define the **size** of N as $k(N) = \sum_{e \in N} k(e)$. We say that N is **extreme** if it has maximum weight among all (l, u) -matchings of size $k(N)$ in G' . A matching M of G is said to be **extreme** if it has maximum weight among all matchings of size $|M|$ in G' .

Algorithm 1 for computing a maximum weight square-free 2-matching differs from the variant for computing a maximum (size) 2-matching only in the fact that we compute a minimum M' -augmenting path instead of a shortest M' -augmenting one. Finding a minimum M' -augmenting path requires computing an M' -augmenting path P with minimum alternating weight. To be able to do this, we need to know that there are no negative cycles in G' . We prove the absence of negative cycles in G' as well as the optimality of the 2-matching computed by Algorithm 1 by using linear programming.

The weighted square-free 2-matching problem can be formulated as an integer program as follows. We assign a variable $x(e)$ for each edge $e \in E$. Any such variable can take on only two values: 0 or 1, where setting a variable $x(e)$ to 1 denotes including e in the 2-matching. To ensure that variables $x(e)$ encode a 2-matching we add constraint 2b. Constraint 2c means that for any square s of the graph at most three of its edges can belong to the 2-matching.

Let S denote the set of all squares of G and $x \in \mathbb{R}^{E(G)}$. The weighted square-free 2-matching problem can be formulated as an integer program, whose linear programming relaxation is the following:

$$(P) \quad \text{maximize} \quad \sum_{e \in E(G)} w(e)x(e) \tag{1a}$$

$$\text{subject to} \quad \sum_{e \in \delta(v)} x(e) \leq 2 \quad (\forall v \in V(G)), \tag{1b}$$

$$\sum_{e \in E(s)} x(e) \leq 3 \quad (\forall s \in S), \tag{1c}$$

$$\sum_{e \in E(G)} x(e) = k. \tag{1d}$$

Let x be any feasible solution of (P) and $M = \{e \in E : x(e) = 1\}$. We can check that M is a square-free 2-matching of G . Namely, the first constraint ensures that for any vertex v at most two edges of M are incident to v and the second constraint implies that for any square s of the graph at most three of its edges belong to M . The linear program (P) has been shown to have an integral optimal solution by Makai [24]. For our purposes we need linear programs, which are relaxations of integer programs for, correspondingly, the extreme size k square-free 2-matching problem and the extreme size k (l, u) -matching problem. These linear programs (P_k) and (P'_k) and their duals $(D_k), (D'_k)$ are given below, where $x' \in \mathbb{R}^{E'(G)}$.

$$(P_k) \quad \text{maximize} \quad \sum_{e \in E(G)} w(e)x(e) \quad (2a)$$

$$\text{subject to} \quad \sum_{e \in \delta(v)} x(e) \leq 2 \quad (\forall v \in V(G)), \quad (2b)$$

$$\sum_{e \in E(s)} x(e) \leq 3 \quad (\forall s \in S), \quad (2c)$$

$$\sum_{e \in E(G)} x(e) = k, \quad (2d)$$

$$0 \leq x(e) \leq 1 \quad (\forall e \in E(G)). \quad (2e)$$

$$(D_k) \quad \text{minimize} \quad 2 \sum_{v \in V(G)} p(v) + \sum_{e \in E(G)} q(e) + 3 \sum_{s \in S} \alpha(s) + \beta k \quad (3a)$$

$$\text{subject to} \quad p(u) + p(v) + q(e) + \sum_{s \in S: e \in E(s)} \alpha(s) + \beta \geq w(e) \quad (\forall e = (u, v) \in E(G)), \quad (3b)$$

$$p, q, \alpha \geq 0. \quad (3c)$$

$$(P'_k) \quad \text{maximize} \quad \sum_{e \in E(G)} w'(e)x(e) \quad (4a)$$

$$\text{subject to} \quad \sum_{e \in \delta(v)} x(e) \leq 2 \quad (\forall v \in V(G)), \quad (4b)$$

$$\sum_{e \in \delta(v)} x(e) = 1 \quad (\forall v \in V(G') \setminus V(G)), \quad (4c)$$

$$\sum_{e \in E(G')} k(e)x(e) = k, \quad (4d)$$

$$0 \leq x(e) \leq 1 \quad (\forall e \in E(G')). \quad (4e)$$

$$(D'_k) \quad \text{minimize} \quad 2 \sum_{v \in V(G)} p(v) + \sum_{v \in V(G') \setminus V(G)} p(v) + \sum_{e \in E(G')} q(e) + \beta k \quad (5a)$$

$$\text{subject to} \quad p(u) + p(v) + q(e) + \beta k(e) \geq w'(e) \quad (\forall e = (u, v) \in E(G')), \quad (5b)$$

$$p(v) \geq 0 \quad (\forall v \in V(G)), \quad (5c)$$

$$q \geq 0. \quad (5d)$$

We define the linear program (P') as (P'_k) without the inequality 4d. We denote the dual programs of (P) and (P') , respectively, as (D) and (D') , correspondingly. These dual programs differ from (D_k) and (D'_k) in that they do not contain the variable β .

► **Fact 5.3.** *Consider an optimal integral primal solution x^* of (P_k) and an optimal dual solution $p^*, q^*, \alpha^*, \beta^*$ of (D_k) . Define $M^* = \{e \in E : x^*(e) = 1\}$. From complementarity slackness we have the following:*

73:12 Restricted t -Matchings via Half-Edges

$$e \in M^* \implies p^*(u) + p^*(v) + q^*(e) + \sum_{s \in S: e \in E(s)} \alpha^*(s) + \beta^* = w(e)$$

$$(\forall e = (u, v) \in E(G))$$

$$\begin{aligned} v \text{ is deficient in } M^* &\implies p^*(v) = 0 & (\forall v \in V), \\ e \notin M^* &\implies q^*(e) = 0 & (\forall e \in E), \\ s \text{ is not saturated in } M^* &\implies \alpha^*(s) = 0 & (\forall s \in S). \end{aligned}$$

Similar constraints hold for the other linear programs. We identify a 2-matching with its incidence vector x .

Let us now explain how we use these linear programs. Observe that to show that G' contains no negative cycles, it suffices to demonstrate that M' is extreme in G' , or in other words, that M' is an optimal solution of (P'_k) . Below in Lemma 5.6 we prove that M' is extreme in G' if M is extreme in G . This means that we need to show that for every k , $0 \leq k \leq n$ in iteration k of Algorithm 1, the computed 2-matching M of size k is an optimal solution of (P_k) . Of course, the empty 2-matching is an optimal solution of (P_0) . Assuming that we have an extreme 2-matching of size $k-1$ in M , we build M' and G' and find a minimum M' -augmenting path in G' . Next we show that by applying P to M' we obtain an (l, u) -matching N of size k , which is extreme in G' . This (l, u) -matching N corresponds to a 2-matching M_1 of size k . We prove that the optimality of the solution N of (P'_k) implies the optimality of the solution M_1 of (P_k) .

► **Lemma 5.4.** *Consider any bipartite graph H and $l, u : V(H) \rightarrow \mathbb{N}_{\geq 0}$. Then an (l, u) -matching polytope is defined by the following inequalities:*

$$\begin{aligned} l(v) &\leq x(\delta(v)) \leq u(v) & (\forall v \in V(H)), \\ 0 &\leq x(e) \leq 1 & (\forall e \in E(H)). \end{aligned}$$

Proof. It is known that an incidence matrix of a bipartite graph is totally unimodular, hence incidence matrix A_H of H is totally unimodular. Observe that $P = \{x \in \mathbb{R}^{E(H)} : l \leq A_H x \leq u \wedge 0 \leq x \leq 1\}$, hence P is integral from theory of totally unimodular matrices. ◀

► **Lemma 5.5.** *Linear programs (P'_k) and (P') have integral optimal solutions.*

To compute a minimum M' -augmenting path in G' , we first find an M' -augmenting path P with minimum alternating weight. To be able to do this, we need to know that there are no negative cycles in G' . In the following lemma we prove the absence of negative cycles in G' . Next, if needed, we shorten P .

► **Lemma 5.6.** *Assume that M is an optimal solution to (P_k) . Then M' is an optimal solution to (P'_k) , and thus M' is extreme in G' . Hence, there are no negative cycles in G' .*

► **Lemma 5.7.** *Let M' be an extreme (l, u) -matching in G' and P a minimum M' -augmenting path. Then $N' = M' \oplus P$ is extreme in G' .*

The proof is almost identical to that of Theorem 17.2 in [32].

► **Lemma 5.8.** *Let P be a minimum M' -augmenting path and let $M'' = M' \oplus P$ and $N = \text{img}(M'')$.*

If $M' \oplus P$ is extreme in G' , then N is an optimal solution of (P_k) .

► **Lemma 5.9.** *Assume that M' is a maximum-weight (l, u) -matching of G' . Then M is a maximum-weight square-free 2-matching of G .*

► **Lemma 5.10.** *Let P be a minimum M' -augmenting path and let $M'' = M' \oplus P$ and $M_1 = \text{img}(M'')$. Then $w(M_1) = w'(M') - \tilde{w}'(P)$ and M_1 can contain only unproblematic squares. M_1 can be transformed into a square-free 2-matching M_2 such that $w(M_2) = w(M_1)$.*

Proof. At the beginning we show that $w(M_1) = w'(M'')$. We have that $w'(M'') = w'(M') - \tilde{w}'(P)$. Observe that the flipping of half-edges does not change the weight of M_1 . Hence $w(M_1) = w'(M'')$.

We observe that Fact 4.1 is still valid in the weighted case, because G' contains no negative alternating cycles. The same is true for Observation 4.6 because of Fact 4.1 and the following. We can shorten P going through d and u_s^2, b, c so that it uses three edges $(d, v_c^d), (v_c^d, v_d^c), (v_d^c, c)$ instead, because the weight of each of these two subpaths is the same and equal to $r(c) + r(d)$.

Next we notice that the proof of Lemma 4.7 goes through for the weighted setting as long as Lemma 4.2 is still valid. We now argue that it indeed is. It suffices to prove that if $s = (a, b, c, d)$ is such that $(a, b), (c, d) \in P \setminus M$, $(b, c), (a, d) \in M \setminus P$, then P can be shortened. Suppose that R is a subpath of P that consists of edges strictly between (a, b) and (c, d) and that a and d are its endpoints. (The case that b and c are the endpoints of R is symmetrical.) Let P_R be a path obtained from P by replacing its subpath R by an edge (a, d) . We show that $w'(P_R) \leq w'(P)$, contradicting the choice of P . w is vertex-induced on s , therefore, $w'(P) - w'(P_R) = w'(R) - w'(a, d) = w'(R) + w(a, d) = w'(R) + w(a, b) + w(c, d) - w(b, c) = w'(R) + w'(a, b) + w'(b, c) + w'(c, d) = w'(C)$ where C is an alternating cycle of G consisting of R and three edges of s . Recall that $w'(C) \geq 0$ because M is extreme. ◀

The corollary of Lemmas 4.8, 5.9 and 5.10 is:

► **Theorem 5.11.** *Algorithm 1 computes a maximum (resp. maximum weight) square-free 2-matching of G .*

6 Maximum-weight $K_{t,t}$ -free t -matchings in bipartite graphs

In this section we solve the weighted $K_{t,t}$ -free t -matching problem in bipartite graphs. Since the case of $t = 2$ has already been addressed in the previous section, we assume that $t \geq 3$. Also, similarly as in Section 5, we assume that the weight function w is vertex-induced on every $K_{t,t}$ of G . The general scheme of the algorithm for the weighted $K_{t,t}$ -free t -matching problem is similar to Algorithm 1- we give it below.

■ **Algorithm 2** Computing a maximum weight $K_{t,t}$ -free t -matching of a bipartite graph G .

-
- 1: Let M be an empty t -matching of G .
 - 2: Construct an auxiliary bipartite graph $G' = (V', E')$ with $\mathcal{O}(tn)$ vertices and $\mathcal{O}(m)$ edges, and its t -matching M' of size $\mathcal{O}(tn)$ by replacing some $K_{t,t}$'s of G with gadgets containing half-edges.
 - 3: Compute a minimum M' -augmenting path P of G' . If G' contains no M' -augmenting path with negative alternating weight, stop the algorithm and return M .
 - 4: Apply P to M' obtaining M'' and extract a $K_{t,t}$ -free t -matching M_1 of G from M'' such that $|M_1| = |M| + 1$ and $w(M_1) > w(M)$. Set M as M_1 and go to 2.
-

▷ **Claim 6.1.** Algorithm 2 runs in time $\mathcal{O}(tnm + t^2 n^2 \log n)$.

73:14 Restricted t -Matchings via Half-Edges

Proof. It is possible to implement all steps of an Algorithm 2 except 3 in linear time. Every step is executed $\mathcal{O}(tn)$ times since $|M|$ increases by one every time 4 is executed. \triangleleft

Let us also remark that in the unweighted version of the problem the runtime of Algorithm 2 is $\mathcal{O}(nm + tn^2 + \sqrt{tnm})$ because 3 can be implemented to run in linear time and the algorithm can start not from an empty matching but from a maximum $(t-1)$ -matching, whose computation takes $\mathcal{O}(\sqrt{tnm})$ time. Additionally, we may want to forbid only some subgraphs of G isomorphic to $K_{t,t}$. Then we proceed analogously, but replace only forbidden subgraphs with gadgets.

► **Definition 6.2.** A subgraph H of G isomorphic to $K_{t,t}$ is **saturated** if it contains exactly one non- M -edge.

► **Lemma 6.3.** Any two different saturated $K_{t,t}$'s of G are vertex-disjoint.

Proof. Let H be any saturated $K_{t,t}$ of G . We say that a vertex v of H is **basic** in H if v is an endpoint of the only non- M -edge of H . Otherwise, we say that v is **nonbasic** in H . Let $V_1(G) \cup V_2(G)$ denote the bipartition of $V(G)$. Thus H has exactly two basic vertices: one in $V_1(H)$ and the other in $V_2(H)$.

Let H_1 and H_2 be any two different saturated $K_{t,t}$'s of G with a common vertex $v \in V_1(G_1)$. We first show that it cannot happen that v is nonbasic both in H_1 and H_2 . Suppose to the contrary that v is nonbasic both in H_1 and H_2 . Then all t edges of M incident to v_1 belong both to H_1 and H_2 . Hence, we get that $V_2(H_1) = V_2(H_2)$. Since $t \geq 3$, at least one of the vertices of $V_2(H_1)$ is nonbasic both in H_1 and H_2 , which implies that $V_1(H_1) = V_1(H_2)$, but this contradicts the fact that H_1 and H_2 are different.

Suppose next that v is basic both in H_1 and H_2 . Then v has $t-1$ incident edges of M in H_1 and $t-1$ incident edges of M in H_2 . Since M is a t -matching, at least one of these edges, say (v, v') , belongs both to H_1 and H_2 . This however means that v' is nonbasic both in H_1 and H_2 (because the endpoints of an M -edge cannot be both basic in the same saturated $K_{t,t}$).

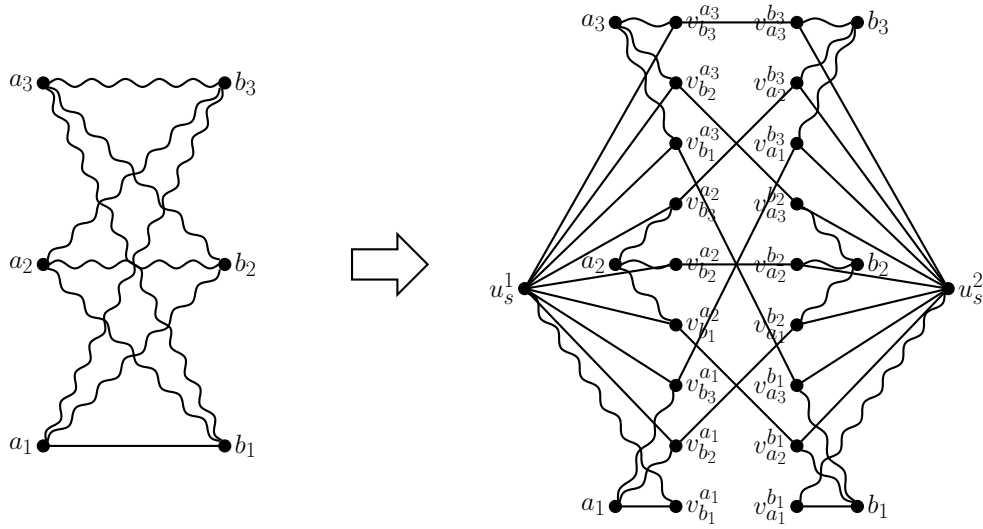
Finally, consider the case when v is basic in H_1 and nonbasic in H_2 . It means that at least two M -edges incident to v , say (v, v') and (v, v'') , belong to both H_1 and H_2 . Vertices v', v'' belong to $V_2(H_2)$, none of them is basic in H_1 and at most one is basic in H_2 . Therefore, at least one of them is nonbasic both in H_1 and H_2 - a contradiction. \triangleleft

► **Observation 6.4.** All saturated $K_{t,t}$'s of G can be found in linear time.

Proof. We can use a linear time algorithm by Galil and Italiano [10]. \triangleleft

We replace every saturated $K_{t,t}$ of G with a gadget described below. By Lemma 6.3, all saturated $K_{t,t}$'s of G are vertex-disjoint.

The construction of the gadget for a saturated $K_{t,t}$ is the following. Let H be any $K_{t,t}$ of G , $A_H = \{a_1, a_2, \dots, a_t\}$ be a set of vertices of one side of H and $B_H = \{b_1, b_2, \dots, b_t\}$ - of the other side. Let $r : V(H) \rightarrow \mathbb{R}$ be a potential function of H . Assume that $(a_1, b_1) \notin M$. For every edge (a_i, b_j) of H , we introduce two subdivision vertices, $v_{b_j}^{a_i}$, $v_{a_i}^{b_j}$, two half-edges and one eliminator. We remove an eliminator of (a_1, b_1) . We set the weight of every half-edge incident to $v \in V(H)$ to $r(v)$. Additionally, we add u_s^1 and u_s^2 to G' . We connect u_s^1 to every subdivision vertex adjacent to some vertex of A_H and we connect u_s^2 to every subdivision vertex adjacent to some vertex of B_H . We add all half-edges of this gadget except for $(a_1, v_{b_1}^{a_1})$ and $(b_1, v_{a_1}^{b_1})$ to M' . Additionally, we add $(u_s^1, v_{b_1}^{a_1})$ and $(u_s^2, v_{a_1}^{b_1})$ to M' .



■ **Figure 3** A gadget for a problematic $K_{3,3}$.

7 Restricted matchings

In this section we consider the following variant of the Restricted Matching Problem. We are given (1) a (not necessarily bipartite) graph $G = (V, E)$, (2) a natural number k , (3) a set of k subsets of edges E_1, E_2, \dots, E_k such that for each $1 \leq i \leq k$ (i) the graph G contains a clique or a bipartite clique on all endpoints of E_i ; in the case of a bipartite clique, it contains the whole set E_i and (ii) the sets E_1, \dots, E_k are almost vertex-disjoint - the endpoints of any two different cliques have at most one vertex in common and (4) k natural numbers r_1, r_2, \dots, r_k . The task is to find a maximum size matching M of G among ones that satisfy the condition: for each $1 \leq i \leq k$ it holds that M contains at most r_i edges of E_i . Any matching M of G that satisfies: for each $1 \leq i \leq k$ $|M \cap E_i| \leq r_i$ is called a **restricted matching**.

To solve this problem we construct a graph G' with gadgets for each of the sets E_i . The construction of G' is similar to the one used for square-free 2-matchings. The precise construction of G' is the following. We start off with $G' = G$. For each $1 \leq i \leq k$ we build a subgraph, called a **gadget for E_i** . Let $n_i = |E_i|$. Each edge (p, q) of E_i is replaced with three new edges $(p, v_q^p), (v_q^p, v_p^q), (v_p^q, q)$, two of which are half-edges of (p, q) and the third one the eliminator of (p, q) . If G contains a clique on all endpoints of E_i we introduce one new global vertex u_i and connect it with every subdivision vertex of an edge belonging to E_i . We set the interval of u_i as $[2n_i - 2r_i, 2n_i - 2r_i]$. If G contains a bipartite clique $K_i = (A_i \cup B_i, E_i)$ on all endpoints of E_i we introduce two new global vertices u_i^1 and u_i^2 . We connect u_i^1 with every subdivision vertex of an edge of E_i , which is a neighbour of a vertex of A_i and similarly, we connect u_i^2 with every subdivision vertex of an edge of E_i , which is a neighbour of a vertex of B_i . We set the interval of both u_i^1 and u_i^2 as $[n_i - r_i, n_i - r_i]$. Let $N = \sum_{i=1}^k n_i$, $R = \sum_{i=1}^k r_i$ and $E_R = \bigcup_{i=1}^k E_i$.

An (l, u) -matching M' of G' is to represent roughly a restricted matching M of G . If M' contains both half-edges of some edge $e \in E_R$, then e is included in M . If M' contains an eliminator of e , then e does not belong to M (is excluded from M). The intuition behind the gadget for the set E_i is that the global vertex or vertices in it are required to be matched to $2n_i - 2r_i$ subdivision vertices of edges of E_i . In this way they block $2n_i - 2r_i$ half-edges, which means that at most $2n_i - (2n_i - 2r_i) = 2r_i$ half-edges of edges of E_i can be present in M' . This implies that at most r_i edges of E_i can appear in the matching M .

► **Theorem 7.1.** *Any maximum (l, u) -matching M' of G' yields a maximum restricted matching of G .*

Proof. Any restricted matching M of G corresponds to an (l, u) -matching M_1 of G' such that $|M_1| = |M| + 2N - R$. To construct such M_1 we proceed as follows. We set M_1 to an empty (l, u) -matching. For each $e \in M \cap E_R$, we add both half-edges of e to M_1 . For each edge $e \in M \setminus E_R$, we add e to M_1 . Next for each $1 \leq i \leq k$ there exist at least $n_i - r_i$ edges of E_i that do not belong to M . We choose any such $(n_i - r_i)$ -element subset $F_i \subseteq E_i$ and for each $e \in F_i$ we connect in M_1 the global vertex/the global vertices $u_i/u_i^1, u_i^2$ to the two subdivision vertices of e . For every edge e of $E_i \setminus (M \cup F_i)$ we add the eliminator of e to M_1 . Let us note that the size of any (l, u) -matching N' of G' satisfies: $2|N'| = \sum_{v \in V} \deg_{N'}(v) + \sum_{v \in V' \setminus V} \deg_{N'}(v) = \sum_{v \in V} \deg_{N'}(v) + 4N - 2R$. This means that the thus constructed M_1 has size $|M| + 2N - R$.

Consider now any (l, u) -matching M' of G' . If for every edge $e \in E_r$ it holds that either both half-edges of e are contained in M' or none, then we say that M' is *integral*. Every integral M' yields a restricted matching M of G such that $|M| = |M'| - 2N + R$. We next show that even when M' is not integral, we are able to build a restricted matching M such that $|M| = |M'| - 2N + R$. We only need to say what to do with half-edges, i.e., with those edges of E_r for which M' contains only one of their half-edges. We deal with each set E_i separately. Suppose first that E_i is such that the graph G contains a clique on all endpoints of E_i . Let us notice that the number of edges of E_i with exactly one of its half-edges contained in M' is even. Let F_i denote such edges and V'_i denote those endpoints of edges of F_i that are incident to some half-edge of an edge of F_i . Then $|V'_i|$ is even. We pair vertices of V'_i in an arbitrary way and for two vertices u, u' belonging to one pair we replace two half-edges of M' incident with u and u' with one edge (u, u') of M . The case when G contains a bipartite clique on all endpoints of E_i is analogous. ◀

References

- 1 Maxim Babenko. Improved algorithms for even factors and square-free simple b-matchings. *Algorithmica*, 64(3):362–383, 2012.
- 2 Kristóf Bérczi and Yusuke Kobayashi. An algorithm for $(n - 3)$ -connectivity augmentation problem: Jump system approach. *Journal of Combinatorial Theory, Series B*, 102(3):565–587, 2012.
- 3 Kristóf Bérczi and László Végh. Restricted b-matchings in degree-bounded graphs. In *Integer Programming and Combinatorial Optimization*, pages 43–56, 2010.
- 4 Gérard Cornuéjols and William Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29(2):135–159, 1980.
- 5 Marshall Fisher, George Nemhauser, and Laurence Wolsey. An analysis of approximations for finding a maximum weight hamiltonian circuit. *Operations Research*, 27(4):799–809, 1979.
- 6 Tamás Fleiner. Uncrossing a family of set-pairs. *Combinatorica*, 21:145–150, 2001.
- 7 András Frank. Restricted t -matchings in bipartite graphs. *Discrete Applied Mathematics*, 131(2):337–346, 2003.
- 8 András Frank and Tibor Jordán. Minimal edge-coverings of pairs of sets. *Journal of Combinatorial Theory, Series B*, 65(1):73–110, 1995.
- 9 Michael Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.
- 10 Zvi Galil and Giuseppe Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22:57–61, 1991.
- 11 Michael Garey and David Johnson. *Computers and Intractability*. Freeman, 1979.

- 12 Jim Geelen. The C_6 -free 2-factor problem in bipartite graphs is NP-complete. Unpublished, 1999.
- 13 David Hartvigsen. *Extensions of Matching Theory*. PhD thesis, Carnegie-Mellon University, 1984.
- 14 David Hartvigsen. Finding maximum square-free 2-matchings in bipartite graphs. *Journal of Combinatorial Theory, Series B*, 96(5):693–705, 2006.
- 15 David Hartvigsen and Yanzun Li. Maximum cardinality simple 2-matchings in subcubic graphs. *SIAM Journal on Optimization*, 21(3):1027–1045, 2011.
- 16 David Hartvigsen and Yanzun Li. Polyhedron of triangle-free simple 2-matchings in subcubic graphs. *Mathematical Programming*, 138:43–82, 2013.
- 17 Alon Itai, Michael Rodeh, and Steven Tanimoto. Some matching problems for bipartite graphs. *Journal of the ACM*, 25(4):517–525, 1978.
- 18 Zoltán Király. C_4 -free 2-factors in bipartite graphs. Technical report, Egerváry Research Group, 1999.
- 19 Zoltán Király. Restricted t -matchings in bipartite graphs. Technical report, Egerváry Research Group, 2009.
- 20 Yusuke Kobayashi. A simple algorithm for finding a maximum triangle-free 2-matching in subcubic graphs. *Discrete Optimization*, 7:197–202, 2010.
- 21 Yusuke Kobayashi. Weighted triangle-free 2-matching problem with edge-disjoint forbidden triangles. In *Integer Programming and Combinatorial Optimization*, pages 280–293, 2020.
- 22 Yusuke Kobayashi and Xin Yin. An algorithm for finding a maximum t -matching excluding complete partite subgraphs. *Discrete Optimization*, 9(2):98–108, 2012.
- 23 László Lovász and Michael Plummer. *Matching theory*. AMS Chelsea Publishing, corrected reprint of the 1986 original edition, 2009.
- 24 Márton Makai. On maximum cost $K_{t,t}$ -free t -matchings of bipartite graphs. *SIAM Journal on Discrete Mathematics*, 21:349–360, 2007.
- 25 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *Combinatorial Optimization*, pages 344–355, 2012.
- 26 Yunsun Nam. *Matching Theory: Subgraphs with Degree Constraints and other Properties*. PhD thesis, University of British Columbia, 1994.
- 27 Katarzyna Paluch, Khaled Elbassioni, and Anke van Zuylen. Simpler approximation of the maximum asymmetric traveling salesman problem. In *29th International Symposium on Theoretical Aspects of Computer Science*, pages 501–506, 2012.
- 28 Katarzyna Paluch and Mateusz Wasylkiewicz. A simple combinatorial algorithm for restricted 2-matchings in subcubic graphs - via half-edges. *Information Processing Letters*, 171, 2021.
- 29 Gyula Pap. Alternating paths revisited ii: restricted b -matchings in bipartite graphs. Technical report, Egerváry Research Group, 2005.
- 30 Gyula Pap. Combinatorial algorithms for matchings, even factors and square-free 2-factors. *Mathematical Programming*, 110:57–69, 2007.
- 31 Irena Rusu. Maximum weight edge-constrained matchings. *Discrete Applied Mathematics*, 156(5):662–672, 2008.
- 32 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.
- 33 Kenjiro Takazawa. A weighted $K_{t,t}$ -free t -factor algorithm for bipartite graphs. *Mathematics of Operations Research*, 34(2):351–362, 2009.
- 34 Kenjiro Takazawa. Decomposition theorems for square-free 2-matchings in bipartite graphs. *Discrete Applied Mathematics*, 233:215–223, 2017.
- 35 Kenjiro Takazawa. Finding a maximum 2-matching excluding prescribed cycles in bipartite graphs. *Discrete Optimization*, 26:26–40, 2017.
- 36 László Végh and András Benczúr. Primal-dual approach for directed vertex connectivity augmentation and generalizations. In *SODA 2005. Proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms.*, pages 186–194, 2005.

Beating Random Assignment for Approximating Quantum 2-Local Hamiltonian Problems

Ojas Parekh ✉

Sandia National Laboratories, Albuquerque, NM, USA

Kevin Thompson ✉

Sandia National Laboratories, Albuquerque, NM, USA

Abstract

The quantum k -Local Hamiltonian problem is a natural generalization of classical constraint satisfaction problems (k -CSP) and is complete for QMA, a quantum analog of NP. Although the complexity of k -Local Hamiltonian problems has been well studied, only a handful of approximation results are known. For Max 2-Local Hamiltonian where each term is a rank 3 projector, a natural quantum generalization of classical Max 2-SAT, the best known approximation algorithm was the trivial random assignment, yielding a 0.75-approximation. We present the first approximation algorithm beating this bound, a classical polynomial-time 0.764-approximation. For strictly quadratic instances, which are maximally entangled instances, we provide a 0.801 approximation algorithm, and numerically demonstrate that our algorithm is likely a 0.821-approximation. We conjecture these are the hardest instances to approximate. We also give improved approximations for quantum generalizations of other related classical 2-CSPs. Finally, we exploit quantum connections to a generalization of the Grothendieck problem to obtain a classical constant-factor approximation for the physically relevant special case of strictly quadratic traceless 2-Local Hamiltonians on bipartite interaction graphs, where a inverse logarithmic approximation was the best previously known (for general interaction graphs). Our work employs recently developed techniques for analyzing classical approximations of CSPs and is intended to be accessible to both quantum information scientists and classical computer scientists.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Semidefinite programming; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Approximation Algorithms, Local Hamiltonian

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.74

Related Version *Full Version:* <https://arxiv.org/abs/2012.12347>

Funding Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Accelerated Research in Quantum Computing and Quantum Algorithms Teams programs.

1 Introduction

The design and analysis of approximation algorithms [37, 38] is an extensively studied area in theoretical computer science. In this setting, we are given some (generally NP-hard) optimization problem, and we are tasked with producing a valid (feasible) solution with objective within some provable factor of the optimal objective value. To understand this formally imagine we are given some optimization problem \mathcal{P} , which corresponds to an infinite set of problem instances $\{P_i\}$. Each problem instance corresponds to a triple $P_i = (f_i, \mathcal{T}_i, n_i)$ where $n_i \in \mathbb{N}_+$, $\mathcal{T}_i \subseteq \{0, 1\}^{n_i}$, and $f_i : \mathcal{T}_i \rightarrow \mathbb{R}_+$ is an objective function. This gives rise to an optimization problem of the form: $OPT_i = \max_{s \in \mathcal{T}_i} f_i(s)$.



© Ojas Parekh and Kevin Thompson;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 74; pp. 74:1–74:18

Leibniz International Proceedings in Informatics



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

An approximation algorithm \mathcal{A} acts on an efficient description of an instance to produce some feasible solution to the problem: $\mathcal{A}(f_i, \mathcal{T}_i, n_i) = \bar{s}_i \in \mathcal{T}_i$ in time polynomial in the instance size (polynomial in n_i). It is said that algorithm has approximation factor α for $0 < \alpha \leq 1$ if in the worst-case (over all instances), the solution produced by the algorithm is a factor of α off of the optimal answer:

$$\min_i \frac{f_i(\bar{s}_i)}{OPT_i} \geq \alpha.$$

Since we should not expect to solve NP-hard problems in polynomial time, an interesting question is then the approximability of NP-hard optimization problems, or the study of which approximation factors α are obtainable for different problems. As one might expect, approximability is highly problem sensitive and there are many classes of natural problems with widely varying approximability [15, 17, 23, 37, 38].

2-Local Hamiltonian. In stark contrast, although QMA-hard quantum optimization problems arise naturally through well-known physically motivated problems [5, 35], they have very few known approximation algorithms with provable approximation factors [2, 4, 8, 11, 18, 19, 21, 22, 31]¹. The QMA-hard optimization studied in these works, as well as the problem we will study here, is the *2-Local Hamiltonian problem* [24, 25].

An instance of this problem is specified by a problem size, n , as well as a set of 2-local interactions (i.e., interactions on pairs of qubits), $\{H_{ij}\}_{ij \in E}$. Each $H_{ij} \in \mathbb{C}^{2^n \times 2^n}$ is some *local* Hamiltonian which can be written as the tensor product of $n - 2$ identity terms with some nontrivial operator, $\mathcal{O}_{ij} \in \mathbb{C}^{4 \times 4}$, that acts on at most 2 qubits, i.e. $H_{ij} = \mathcal{O}_{ij} \otimes \mathbb{I}_{[n] \setminus \{i,j\}} \in \mathbb{C}^{2^n \times 2^n}$ (see Section 2.1 for notation). The optimization problem corresponding to a particular instance is to find the smallest or largest eigenvalue, λ_{\min} or λ_{\max} , of $H = \sum_{ij \in E} H_{ij}$.

► **Problem 1** (QLH: Quantum Max 2-Local Hamiltonian). *Given a problem size, n , as well as a classical description of a set of 2-local terms $\{H_{ij} = \mathcal{O}_{ij} \otimes \mathbb{I}_{[n] \setminus \{i,j\}}\}_{ij \in E}$ with $H_{ij} \in \mathbb{C}^{2^n \times 2^n}$ Hermitian, find:*

$$\lambda_{\max}(H) := \max_{|\phi\rangle \in (\mathbb{C}^2)^{\otimes n}} \langle \phi | H | \phi \rangle = \max_{\substack{\rho \in \mathbb{C}^{2^n \times 2^n} \\ \text{Tr}(\rho)=1, \rho \succeq 0}} \text{Tr}[H\rho], \text{ where } H := \sum_{ij \in E} H_{ij}.$$

Ideally, an algorithm solving this problem would also produce a description of or access to a corresponding eigenvector. We will focus on approximating λ_{\max} for special cases of 2-local Hamiltonian. Although exactly computing $\lambda_{\max}(H)$ is equivalent to computing $\lambda_{\min}(-H)$, the approximability of these problems can differ. An approximation algorithm, \mathcal{A} , acts on the description of the local Hamiltonians $\{H_{ij}\}$ to produce a classical description of a valid quantum state, $\bar{\rho}$. Once again we say that the algorithm achieves approximation factor α if:

$$\frac{\text{Tr}\left[\left(\sum_{ij} H_{ij}\right) \bar{\rho}\right]}{\lambda_{\max}\left(\sum_{ij} H_{ij}\right)} \geq \alpha, \text{ for all instances.}$$

¹ Here and throughout this paper we mean a *classical* algorithm which takes as input a classical description of a quantum problem and produces a classical description of a quantum state. An approximation algorithm for a QMA-hard problem can have several natural meanings distinct from this (quantum input, quantum algorithm which produces classical output, etc.).

Generally we assume some property of the Hamiltonian which forces $\lambda_{\max}(H) > 0$ so that this is a sensible definition. A common assumption [18, 21, 24] is that the terms H_{ij} are positive semi-definite (PSD) and nonzero. We note that when all of the terms H_{ij} are taken to be diagonal projectors (in say, the standard computational basis), the corresponding instance of 2-Local Hamiltonian corresponds precisely to an instance of the classical 2-Constraint-Satisfaction problem (2-CSP). In this case, the 4 diagonal entries of \mathcal{O}_{ij} correspond to the $\{0, 1\}$ output values of a Boolean function on variables x_i and x_j corresponding to i and j . See Appendix E in [30] for more details as well as a classical motivation for 2-Local Hamiltonian. In addition Table 1 highlights classical 2-CSP specializations of quantum Max 2-local Hamiltonian problems for which approximation algorithms are known.

The 2-Local Hamiltonian problem is interesting in many different contexts of physics and quantum information [24, 25, 29]. This problem is manifestly interesting to physicists because the 2-local nature of the problem matches the local nature of many physical systems (spin chains, Ising model, etc.). Hence, the study of eigenstates and energies is of utmost importance, and has been since the beginnings of quantum mechanics itself [6]. From a theoretical computer science perspective, the 2-Local Hamiltonian problem is interesting for the same reasons that classical approximation algorithms are interesting. Under standard complexity theoretic assumptions, we should not expect to be able to efficiently solve the problem, so the interesting direction is the study of the approximability of the problem. Can we find rigorous approximation algorithms, and how well can we expect to be able to approximate the answer? Moreover, which classes of instances admit constant-factor approximation algorithms? Akin to the classical PCP Theorem, the potential inapproximability of local Hamiltonian problems to within constant factors is known as the Quantum PCP Conjecture. The resolution of the conjecture would yield insight into properties of quantum mechanics and entanglement (e.g., Section 1.3 in [1]).

1.1 Our Contributions

The QLH problems we consider generalize a variety of classical optimization problems, including Max 2-SAT, Max Cut, general Max 2-CSP, and the Grothendieck problem (see Table 1 for our results). We focus on QLH where each term H_{ij} is a projector and the special case where each projector is strictly quadratic, both of which remain QMA-hard [32]. The strictly quadratic case precludes non-identity 1-local terms (i.e. $\mathcal{O}_i \otimes \mathbb{I}_{[n] \setminus \{i\}}$ with $\mathcal{O}_i \neq \mathbb{I}$) that may be implicit in a 2-local term (see Definition 8). In this case $\mathcal{O}_{ij} = w_{ij}P_{ij}$, where P_{ij} is a 2-qubit projector, and $w_{ij} \geq 0$ is a weight. There are three interesting cases, depending on the rank of P_{ij} . We will obtain approximation factors for each.

► **Problem 2 (QLHP(k): Quantum Max 2-Local Hamiltonian on Projectors).** *Given a problem size, n , as well as a classical description of a set of 2-local terms $\{H_{ij} = w_{ij}P_{ij} \otimes \mathbb{I}_{[n] \setminus \{i,j\}}\}_{i,j \in E}$ with $w_{ij} \geq 0$ and $P_{ij} \in \mathbb{C}^{4 \times 4}$ a 2-qubit projector of rank at least k , find $\lambda_{\max}(\sum_{i,j \in E} H_{ij})$.*

► **Remark 3.** Although Problem 2 is formulated for a single term H_{ij} per pair of qubits i, j , our techniques apply when multiple terms are present per pair. Since any $\mathcal{O}_{ij} \succeq 0$ can be written as a positive combination of rank-1 projectors, the version of QLHP(1) we solve more generally captures instances of QLH where each $H_{ij} \succeq 0$.

► **Theorem 4 (Informal).** *Given an instance of QLHP(k), $\{H_{ij} = w_{ij}P_{ij} \otimes \mathbb{I}_{[n] \setminus \{i,j\}}\}$, where all 2-local projectors P_{ij} are rank $k \in \{1, 2, 3\}$, we give a classical randomized polynomial-time algorithm with approximation ratio $\alpha(k)$, where*

$$\alpha(k) = \begin{cases} 0.387 & \text{if } k = 1 \\ 0.565 & \text{if } k = 2 \\ 0.764 & \text{if } k = 3. \end{cases}$$

► **Theorem 5 (Informal).** *If in addition to the assumptions of Theorem 4, the terms H_{ij} are strictly quadratic, we give a classical randomized polynomial-time algorithm with approximation ratio $\alpha(k)$, where*

$$\alpha(k) = \begin{cases} 0.467 & \text{if } k = 1 \\ 0.639 & \text{if } k = 2 \\ 0.805 & \text{if } k = 3. \end{cases}$$

The decision version of the problem we consider is known as Quantum-SAT and was introduced in 2006 by Bravyi [10], and the approximability of QLHP was first considered in 2011 by Gharibian and Kempe [18], who observed that the maximally mixed state trivially achieves an approximation ratio of $k/4$ for $k \in \{1, 2, 3\}$. The only nontrivial result previously known is a 0.328-approximation for the $k = 1$ case by Hallgren, Lee, and Parekh [21]. In contrast to previous works [19, 21], we are able to directly analyze the expected performance of our algorithm rather than appealing to known but weaker black-boxes.

Significance of our work. We give the first approximation algorithms beating random assignment for QLHP. We show how to move beyond numerical evaluation of approximation ratios for QLH, which is not as critical in the classical case that enjoys only a handful of parameters. This is accomplished by: (1) reducing the number of parameters for analysis of a single term from 18 to 3 (in the strictly quadratic case), and (2) explicitly computing the coefficients of a Hermite decomposition of a multivariate Gaussian expectation. The latter generalizes previous results of Briët, de Oliveira Filho, and Vallentin [13] employed in [19]. We are able to analyze a natural generalization of classical hyperplane rounding that we expect will enable approximation of other QLH problems.

Strictly quadratic instances. We believe the strictly quadratic case is an interesting special case for several reasons. As noted, one of the difficulties in analyzing rounding schemes for QLH is the sheer number of parameters involved. The quadratic case reduces the number of parameters to consider, while still including physically relevant QMA-hard instances such as the Max Heisenberg model that serves as a quantum generalization of Max Cut [19]. Indeed we believe that quadratic instances allow one to glean insights and develop techniques that might otherwise be obscured in more general instances. Some of the first rigorous approximation algorithms for QLH that go beyond product states were recently developed for quantum Max Cut [2, 31]. Moreover, maximally entangled instances are strictly quadratic, and we conjecture these are the hardest cases to approximate.

Numerical results and upper bounds. We conjecture that the true performance of our algorithm for general QLHP, including linear terms, is:

► **Conjecture 6.** (Informal) Our rounding algorithm achieves approximation ratio:

$$\alpha(k) = \begin{cases} 0.498 & \text{if } k = 1 \\ 0.653 & \text{if } k = 2 \\ 0.821 & \text{if } k = 3, \end{cases}$$

and these quantities match the worst-case gap between OPT and our SDP upper bound.

We have indeed confirmed the approximation ratios in Conjecture 6 through numerical experiments. The difficulty in taking these encouraging results as fact is: (1) the increase in complexity for an exhaustive search as the number of parameters grows, and (2) although we do give series expansions for the expected performance of the algorithm, establishing smoothness of the expected performance is more difficult than for classical CSP approximation algorithms. Classical 2-CSP approximation factors are also established numerically; however, the performance is usually a well-behaved function of a single parameter, which makes exhaustive numerical search more plausible. There has been limited work on rigorous analysis of 2-CSP approximation guarantees [36]. We note that for the strictly quadratic case, our approach requires a search over only three parameters, rendering these numerical results more plausible. We also note that the performance of a similar approximation algorithm for quantum Max Cut, a special case of QLPH(1), is a hypergeometric function in one parameter, and the corresponding approximation guarantee is indeed 0.498 [19].

An upper bound on $\alpha(k)$ is given below for an instance of QLPH(k) on 2 qubits, with multiple edges that are each strictly quadratic rank- k projectors. These bounds are fairly close to the values in Conjecture 6.

► **Theorem 7.** There exist an instance of QLH on a single edge e , where $H_e \succeq 0$ is strictly quadratic, is a convex combination of rank- k projectors, and satisfies:

$$\max_{|\phi_1\rangle \in \mathbb{C}^2, |\phi_2\rangle \in \mathbb{C}^2} \langle \phi_1 | \otimes \langle \phi_2 | H_e | \phi_1 \rangle \otimes | \phi_2 \rangle \leq \beta(k) \cdot \lambda_{\max}(H_e),$$

$$\text{where } \beta(k) = \begin{cases} 1/2 & \text{if } k = 1 \\ 2/3 & \text{if } k = 2 \\ 5/6 & \text{if } k = 3. \end{cases}$$

Proof. The Bell states take their usual definition:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}, \quad |\Phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}, \quad |\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}, \quad \text{and } |\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}.$$

We define,

$$H_e(k) = \frac{k-1}{3} \mathbb{I} + \frac{4-k}{3} |\Psi^-\rangle \langle \Psi^-|.$$

Let $\mu(k) = \max_{|\phi_1\rangle \in \mathbb{C}^2, |\phi_2\rangle \in \mathbb{C}^2} \langle \phi_1 | \otimes \langle \phi_2 | H_e(k) | \phi_1 \rangle \otimes | \phi_2 \rangle$. From, e.g., [19], we know that $\mu(1) = 1/2$, while $\lambda_{\max}(H_e(1)) = 1$ since $|\Psi^-\rangle$ is an eigenvector of the rank-1 projector $H_e(1)$. For $k > 1$ we have that

$$\begin{aligned} \mu(k) &= \frac{k-1}{3} + \frac{4-k}{3} \mu(1) = \frac{k+2}{6}, \text{ and} \\ \lambda_{\max}(H_e(k)) &= \frac{k-1}{3} + \frac{4-k}{3} \lambda_{\max}(H_e(1)) = 1, \end{aligned}$$

yielding the desired values of $\beta(k)$. We give explicit descriptions of $H_e(k)$ as convex combinations of rank- k projectors for $k = 2, 3$:

$$\begin{aligned} H_e(2) &= \frac{1}{3}(|\Psi^-\rangle\langle\Psi^-| + |\Phi^+\rangle\langle\Phi^+|) + \frac{1}{3}(|\Psi^-\rangle\langle\Psi^-| + |\Phi^-\rangle\langle\Phi^-|) \\ &\quad + \frac{1}{3}(|\Psi^-\rangle\langle\Psi^-| + |\Psi^+\rangle\langle\Psi^+|), \text{ and} \\ H_e(3) &= \frac{1}{3}(\mathbb{I} - |\Phi^+\rangle\langle\Phi^+|) + \frac{1}{3}(\mathbb{I} - |\Phi^-\rangle\langle\Phi^-|) + \frac{1}{3}(\mathbb{I} - |\Psi^+\rangle\langle\Psi^+|). \end{aligned}$$

1.2 Related Work

In the interest of classical approximations of a 2-Local Hamiltonian instance H , let $OPT := \lambda_{\max}(H)$ and define

$$OPT_{\text{prod}} := \max_{\substack{|\phi_1\rangle, \dots, |\phi_n\rangle \\ \in \mathbb{C}^2}} \langle \phi_1 | \otimes \dots \otimes \langle \phi_n | H | \phi_1 \rangle \otimes \dots \otimes | \phi_n \rangle$$

to be the product state² with the largest objective value or *energy*.

Approximations for QLH generally make assumptions on the form of the terms H_{ij} . One common assumption is on the *geometry* of the interactions in E . Bansal, Bravyi, and Terhal show that 2-Local Hamiltonian on bounded-degree planar graphs admits a polynomial-time approximation scheme³ (PTAS) [4], and Brandão and Harrow generalize this to arbitrary planar graphs [8]. On the other end, for k -Local Hamiltonian on dense graphs, Gharibian and Kempe give a PTAS with respect to OPT_{prod} [18], and Brandão and Harrow extend this result to obtain a PTAS for dense graphs with respect to OPT [8]. Brandão and Harrow also show the existence of product-states with energy nearly that of OPT or give product-state approximations for a variety of graph classes [8].

Bravyi, Gosset, König, and Temme give an $\Omega(\frac{1}{\log n})$ -approximation for traceless⁴ QLH [11]. This case is general enough to capture classical problems with no constant-factor approximations [3]. Harrow and Montanaro give an approximation algorithm for traceless k -Local Hamiltonian with respect to the maximum degree and size of the interaction hypergraph [22]. Note that approximating traceless QLH generalizes all problems considered in this paper, since adding copies of the identity only improves the approximation factor; however there is no reason to expect such analysis could be used to prove constant factor approximations for the classes we study.

A unifying theme among recent approaches (see Table 1 for approximation guarantees) is employing a semi-definite program (SDP) to provide an upper bound on OPT and then using generalization of some classical randomized rounding scheme to produce a product state [11, 19, 21]. Such an approach was first carried out by Brandão and Harrow [8]. Gharibian and Parekh [19] consider a QMA-hard rank-1 QLHP problem that is a generalization of the classical Max Cut problem. Hallgren, Lee, and Parekh [21] study QLHP and give the first approximation beating random assignment for rank-1 QLHP. They also provide

² As is suggested by the expression, a product state is a quantum state which factors according to tensor product of individual quantum states. Such states have no entanglement and are considered “classical” states.

³ This is an approximation algorithm that for a constant $\varepsilon > 0$, allows a $1-\varepsilon$ approximation factor at the expense of a runtime that depends on $1/\varepsilon$.

⁴ A traceless instance is one with $\text{Tr}[H] = 0$. Alternatively, when expressed as a polynomial in the Pauli basis, H is traceless if it has no identity term.

approximations when each term is a product term, $H_{ij} = H_i \otimes H_j$, which is a QMA-hard class of QLH. An approximation result of [8] leverages the quantum Lasserre hierarchy of SDPs. The first level of this hierarchy yields a natural SDP relaxation that is employed by [11] and [19]. Our work adopts the approach of [21] and strengthens the natural SDP relaxation with additional constraints related that enforce positivity of 2-qubit marginals. Both [19] and [21] appeal to approximation results of Briët, de Oliveira Filho, and Vallentin [12, 13] to analyze the expected performance of their rounding algorithms. The rounding scheme of [11] is a generalization of a classical approximation by Charikar and Wirth [14].

Beyond product states. Gharibian and Parekh [19] give a 0.498-approximation for their quantum generalization of Max Cut, where $\frac{1}{2}$ is the best possible approximation by product states (see Theorem 7). Anshu, Gosset, and Morenz [2] demonstrate that it is possible to beat a $\frac{1}{2}$ -approximation for quantum Max Cut by furnishing a classical randomized approximation that outputs a description of a tensor product of 1- and 2-qubit states rather than product states, which are tensor products of 1-qubit states. This result does not rely on an SDP as an upper bound on OPT , instead appealing to well-known monogamy of entanglement bounds for the Heisenberg model. Parekh and Thompson [31] observe that a similar type of approximation is possible using the second level of the quantum Lasserre hierarchy and obtain a slight improvement over the approximation ratio of [2].

Although these results may seem to suggest that the future of approximation algorithms for QLH should look beyond product states, a solid understanding of the approximability of product states is likely necessary for *any* type of general approximation algorithm for QLH. Brandão and Harrow [8] show that for (certain generalizations of) regular graphs, OPT_{prod} approaches OPT as the degree increases. Indeed, trading off product-state approximations with more general quantum states is a key ingredient in both [2] and [31].

2 Semidefinite Relaxation and Rounding Approach

In this section we present a rigorous but high-level overview of our approach, with technical lemmas deferred to later sections. We define the main problems considered and our semidefinite relaxation and rounding algorithm. We conclude by motivating the analysis that will occur in detail in the full version of this paper, [30].

2.1 Preliminaries

Quantum information notation. We adopt some standard notations used in quantum information [28]. The *kets* $|0\rangle := [0, 1]$ and $|1\rangle := [1, 0]$ represent the standard basis vectors for \mathbb{C}^2 , while the *bras* $\langle 0|$ and $\langle 1|$ represent their conjugate transposes. The $d \times d$ identity matrix is denoted by \mathbb{I}_d , and the subscript will be omitted when redundant. We obtain the standard bases for \mathbb{C}^{2^n} as $|b_1 b_2 \dots b_n\rangle := |b_1\rangle |b_2\rangle \dots |b_n\rangle := |b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_n\rangle$, with $b_i \in \{0, 1\}$. The Pauli matrices will have the usual definition:

$$\sigma^0 = \mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \sigma^1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma^2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \text{ and } \sigma^3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (1)$$

We will generally use subscripts to indicate quantum subsystems. If ρ is a density matrix on n qubits, for instance, ρ_{ij} will correspond to the marginal density matrix on qubits i and j , i.e. the partial trace $\rho_{ij} = \text{Tr}_{[n] \setminus \{i, j\}}[\rho]$ (e.g. [28], Section 2.4.3). Similarly, σ_i^j corresponds to Pauli j on qubit i . Subscripts will supercede position in many cases in the paper, for instance $\sigma_i^1 \otimes \sigma_j^2 \otimes \mathbb{I}_{[n] \setminus \{i, j\}}$ is meant as $\mathbb{I} \otimes \mathbb{I} \otimes \dots \otimes \sigma^1 \otimes \dots \otimes \mathbb{I} \otimes \sigma^2 \otimes \dots \otimes \mathbb{I}$ where σ^1 is at

■ **Table 1 Approximation algorithms for Max 2-local Hamiltonian problems.** The number of qubits or Boolean variables is n , and the set of pairwise constraints is E . For readability in the table below, we omit weights $w_{ij} \geq 0$ that may be present in both 2-local Hamiltonian (2-LH) and related classical constraint satisfaction (2-CSP) problems. [30] provides more details on the relationship between 2-LH and 2-CSP, as well as definitions for X_i, Y_i, Z_i . An “N” denotes a numerical result; the classical results are implicitly numerical since they are obtained by numerically finding the worst-case ratio over a range of parameters. The abbreviation “quad.” refers to strictly quadratic instances.

Max 2-LH problem (QMA-hard)	Max 2-CSP specialization (NP-hard)	Classical approx. for 2-CSP	Classical approx. for 2-LH problem (product state)
Traceless $\sum_{ij \in E} H_{ij} \otimes \mathbb{I}_{[n] \setminus \{ij\}}$ H_{ij} has no \mathbb{I} terms	Classical Ising $\max -\sum_{ij \in E} z_i z_j$ $z_i \in \{\pm 1\}$	$\Omega(\frac{1}{\log n})$ [14]	$\Omega(\frac{1}{\log n})^\dagger$ [11]
Bipartite Traceless $\sum_{e \in E} H_{ij} \otimes \mathbb{I}_{[n] \setminus \{ij\}}$ H_{ij} has no \mathbb{I} terms E bipartite	Grothendieck $\max -\sum_{ij \in E} z_i z_j$ $z_i \in \{\pm 1\}$ E bipartite	$0.561 + \varepsilon$ [9]	0.187[†](quad.)
Positive/Rank 1 $\sum_{ij \in E} H_{ij} \otimes \mathbb{I}_{[n] \setminus \{ij\}}$ $\mathbb{I} \succeq H_{ij} \succeq 0$ ($\equiv H_{ij}$ rank 1 projector)	Max 2-CSP ($\equiv 1$ satisfying assignment per clause)	0.874 [27]	0.25 (random) 0.328 [21] 0.387 0.467 (quad.) 0.498 (quad., N) 0.5 (upper bound)
Max Heisenberg $\sum_{ij \in E} \mathbb{I} - X_i X_j - Y_i Y_j - Z_i Z_j$ (special case of above)	Max Cut $\max \sum_{ij \in E} 1 - z_i z_j$ $z_i \in \{\pm 1\}$	0.878 [20]	0.25 (random) 0.498 [19] 0.5 (upper bound) 0.53 [*] [2] 0.533 [*] [31]
Rank 2 $\sum_{ij \in E} H_{ij} \otimes \mathbb{I}_{[n] \setminus \{ij\}}$ H_{ij} rank 2 projector	Max 2-CSP with 2 satisfying assignments/clause	0.874 [27]	0.5 (random) 0.565 0.639 (quad.) 0.653 (quad., N) 0.667 (upper bound)
2-QSAT $\sum_{ij \in E} H_{ij} \otimes \mathbb{I}_{[n] \setminus \{ij\}}$ H_{ij} rank 3 projector	Max 2-SAT ($\equiv 3$ satisfying assignments/clause)	0.940 [27]	0.75 (random) 0.764 0.805 (quad.) 0.821 (quad., N) 0.834 (upper bound)

* This exceeds the product-state upper bound because it is achieved by a classical approximation algorithm that rounds to a description of a non-product state.

† For any traceless 2-LH problem, we obtain a product-state approximation ratio that is $\frac{1}{3}$ of an approximation ratio for a related classical CSP, using the appropriate classical approximation algorithm as a black box (see Appendix F in the extended version of this article [30]). This also gives another algorithm and proof for the result of [11] in the first row.

the i th position and σ^2 is at the j th position. We encourage readers familiar with classical constraint satisfaction problems to consult Appendix E in [30], which casts such problems as quantum local Hamiltonian problems.

2-local Hamiltonian. Our approximations for QLHP allow multiedges, i.e. distinct edges e and e' on the same pair of qubits i, j . For the sake of exposition, we generally ignore this possibility and conduct our analysis assuming we have a single term H_{ij} for each $ij \in E$. However, when necessary, we will use the notation e_1 and e_2 to refer to the qubits on which an edge e acts. In this context a term H_e is 2-local if it can be written in the form $H_e = \mathcal{O}_e \otimes \mathbb{I}_{[n] \setminus \{e_1, e_2\}}$. Local Hamiltonians have polynomially-sized descriptions which can be given in terms of the local operators \mathcal{O}_e , but for our purposes the details of the description will not be important. We will use $\text{rank}(H_e)$ to mean $\text{rank}(\mathcal{O}_e)$. The actual rank of H_e is $\text{rank}(\mathcal{O}_e)2^{n-2}$, but for ease of exposition we will say that the “rank” of a 2-local term is equal to the rank of its non-trivial part.

The bulk of our work focuses on strictly quadratic instances of QLHP, allowing us to express our main ideas more clearly. The strictly quadratic case precludes non-identity 1-local terms (i.e. $\mathcal{O}_i \otimes \mathbb{I}_{[n] \setminus \{i\}}$ with $\mathcal{O}_i \neq \mathbb{I}$) that may be implicit in a 2-local term.

► **Definition 8 (Strictly Quadratic).** Let H_e be a 2-local term on n qubits. Write $H_e = \mathcal{O}_e \otimes \mathbb{I}_{[n] \setminus \{e_1, e_2\}}$ for some nontrivial operator \mathcal{O}_e . Express \mathcal{O}_e in the Pauli basis as:

$$\mathcal{O}_e = \sum_{k,l=0}^3 \alpha_{k,l} \sigma^k \otimes \sigma^l. \quad (2)$$

We say that H_e is a strictly quadratic if $\alpha_{k,0} = 0$ for all $k \neq 0$, and $\alpha_{0,l} = 0$ for all $l \neq 0$.

Note that the coefficients in Equation (2) may be obtained as $\alpha_{k,l} = \text{Tr}[(\sigma^k \otimes \sigma^l) \mathcal{O}_e] / 4$ and are real since \mathcal{O}_e is Hermitian.

Example. An example of a strictly quadratic instance of QLH is quantum Max Cut [19], where for all $ij \in E$, $\mathcal{O}_{ij} = \frac{1}{4}(\mathbb{I} - \sigma^1 \otimes \sigma^1 - \sigma^2 \otimes \sigma^2 - \sigma^3 \otimes \sigma^3)$. In this case $\mathcal{O}_{ij} = |\Psi^- \rangle \langle \Psi^-|$, with $|\Psi^- \rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$, so that \mathcal{O}_{ij} has rank 1. In general, any maximally entangled pure state on 2 qubits gives rise to a rank-1 strictly quadratic term, since such states must have maximally mixed reduced density matrices.

2.2 Semidefinite Relaxation

We employ a semidefinite programming relaxation for QLH (Problem 1) that is a refinement of the now standard SDP relaxation that has been used in designing approximation algorithms [8, 11, 19]. Our relaxation is related to one used by Hallgren, Lee, and Parekh [21] and may be viewed as a specialization of noncommutative Lasserre hierarchies proposed for quantum information applications [16, 31, 34].

In this section we assume, for the sake of exposition, that there is a single edge ij on any pair of qubits $i, j \in [n]$; however, the relaxation and rounding algorithm may be readily extended to handle general instances of QLH with multiedges. Suppose we have an instance of QLH on n qubits. The first set of variables in our SDPs will be marginal density matrices $\{\rho_{ij}\}$. Since there are n qubits, there are $\binom{n}{2}$ many of these, and each of them is a 4×4 Hermitian matrix. While we cannot impose global consistency, we can force each ρ_{ij} to be a valid density matrix on its own: $\text{Tr}[\rho_{ij}] = 1$ and $\rho_{ij} \succeq 0$ for all $i, j \in [n]$. We could also explicitly force overlapping marginals to be consistent on single qubit density matrices, however this will be implicit through our use of moment matrices.

Moment matrices. Suppose we have a quantum state on n qubits $|\psi\rangle \in \mathbb{C}^{2^n}$. Consider the 1-local Pauli operators, $\mathcal{M} = \{\mathbb{I}\} \cup \{\sigma_i^k \otimes \mathbb{I}_{[n] \setminus \{i\}} \mid k \in [3], i \in [n]\}$. We apply each of the $3n+1$ Pauli operators $\mathcal{O} \in \mathcal{M}$ on $|\psi\rangle$ to obtain columns of a matrix $V = \{\mathcal{O}|\psi\rangle\}_{\mathcal{O} \in \mathcal{M}} \in \mathbb{C}^{2^n \times (3n+1)}$. We call $M := V^\dagger V \in \mathbb{C}^{(3n+1) \times (3n+1)}$ the *moment matrix* of $|\psi\rangle$ with respect to \mathcal{M} ; note that M is Hermitian and $M \succeq 0$ by construction. The notation $M(\mathcal{O}, \mathcal{P})$ refers to the entry of M at the row and column corresponding to $\mathcal{O}, \mathcal{P} \in \mathcal{M}$ respectively. We have $M(\mathcal{O}, \mathcal{P}) = \langle \psi | \mathcal{O} \mathcal{P} | \psi \rangle$, for all $\mathcal{O}, \mathcal{P} \in \mathcal{M}$, so that M captures all the 2-local Pauli statistics of $|\psi\rangle$. In particular the quantity $\langle \psi | H | \psi \rangle$ is a linear function of the entries of M for a 2-local Hamiltonian H . If we let \mathcal{M}_k consist of all the k -local tensor products of Paulis, instead of just the 1-local ones, the corresponding moment matrix M_k , of size $O(n^k)$ by $O(n^k)$, includes all the $2k$ -local Pauli statistics. We may obtain SDP relaxations for QLH problems by constructing a relaxed $M_k \succeq 0$ that satisfies linear constraints of the form $\text{Tr}[AM_k] = b$, that a true moment matrix would satisfy. This corresponds to the k th level of noncommutative Lasserre hierarchies introduced for quantum information [16, 31, 34]. Our approach relaxes \mathcal{M}_1 and adds additional constraints enforcing positivity of 2-local marginals; the relaxation we obtain sits between the $k = 1$ and $k = 2$ levels of the noncommutative Lasserre hierarchy.

SDP Relaxation. We define a (relaxed) moment matrix M , which will track local statistics of the set of marginals $\{\rho_{ij}\}$. Let M be a symmetric, $(3n+1) \times (3n+1)$ real matrix whose rows and columns correspond to operators in \mathcal{M} . Entries of M will correspond to coefficients of the marginal density matrices $\{\rho_{ij}\}$ in the Pauli basis. We use the notation $M(\sigma_i^k, \sigma_j^l)$ to refer to entries of M for $i, j \in [n]$ and $k, l \in [3]$; in addition we have a row and column of M indexed by \mathbb{I} . We set $M(\sigma_i^k, \sigma_j^l) = \text{Tr}[\sigma^k \otimes \sigma^l \rho_{ij}]$ for $(i, k), (j, l)$ in $[n] \times [3]$. In addition we set $M(\mathbb{I}, \mathbb{I}) = 1$, and $M(\sigma_i^k, \mathbb{I}) = \text{Tr}[\sigma^k \otimes \mathbb{I} \rho_{ij}]$ for all $(i, k) \in [n] \times [3]$ and $j \in [n]$. Note that this constraint forces consistent single-qubit marginals since

$$\text{Tr}_u[\rho_{iu}] = \text{Tr}_v[\rho_{iv}] \Leftrightarrow \text{Tr}[\sigma^l \otimes \mathbb{I} \rho_{iu}] = \text{Tr}[\sigma^l \otimes \mathbb{I} \rho_{iv}] \quad \forall l.$$

Since M contains all local information of $\{\rho_{ij}\}$, we can use M to evaluate the objective of our SDP. In this direction, we will define a weight matrix for each edge $H_{ij} = w_{ij} \mathcal{O}_{ij} \otimes \mathbb{I}_{n \setminus \{i, j\}}$, where $\mathcal{O}_{ij} \in \mathbb{C}^{4 \times 4}$, and w_{ij} is a scalar weight. We define the $(3n+1) \times (3n+1)$ matrix C_{ij} , which contains the coefficients of \mathcal{O}_{ij} in the Pauli basis:

$$\begin{aligned} C_{ij}(\sigma_i^k, \sigma_j^l) &= C_{ij}(\sigma_j^l, \sigma_i^k) = \text{Tr}[\sigma^k \otimes \sigma^l \mathcal{O}_{ij}] / 8 \quad \forall k, l \in [3], \\ C_{ij}(\sigma_i^k, \mathbb{I}) &= C_{ij}(\mathbb{I}, \sigma_i^k) = \text{Tr}[\sigma^k \otimes \mathbb{I} \mathcal{O}_{ij}] / 8 \quad \forall k \in [3], \\ C_{ij}(\sigma_j^l, \mathbb{I}) &= C_{ij}(\mathbb{I}, \sigma_j^l) = \text{Tr}[\mathbb{I} \otimes \sigma^l \mathcal{O}_{ij}] / 8 \quad \forall l \in [3], \end{aligned} \tag{3}$$

and all other entries of C_{ij} are 0. To illustrate application of the matrix C_{ij} , suppose \mathcal{O}_{ij} and the marginal density matrix ρ_{ij} have Pauli decompositions:

$$\mathcal{O}_{ij} = \sum_{k, l=0}^3 \alpha_{kl} \sigma^k \otimes \sigma^l \quad \text{and} \quad \rho_{ij} = \frac{1}{4} \sum_{k, l=0}^3 \beta_{kl} \sigma^k \otimes \sigma^l.$$

Since, for $k, l \geq 0$, $(\sigma^k)^2 = \mathbb{I}$ and $\text{Tr}[\sigma^k \sigma^l] = 0$ when $k \neq l$, the value we gain from edge ij , ignoring the weight w_{ij} , is written as:

$$\text{Tr}[\mathcal{O}_{ij} \rho_{ij}] = \alpha_{00} \beta_{00} + \sum_{\substack{k, l: \\ (k \neq 0) \vee (l \neq 0)}} \alpha_{kl} \beta_{kl} = \frac{\text{Tr}[\mathcal{O}_{ij}]}{4} + \text{Tr}[C_{ij} M]. \tag{4}$$

With these facts in hand, we may finally give the main SDP relaxation in this work:

► **Problem 9.** Given an instance of QLH (Problem 1) on n qubits with local terms $\{H_{ij} = w_{ij}\mathcal{O}_{ij} \otimes \mathbb{I}_{[n]\setminus\{i,j\}}\}$, let C_{ij} be defined according to Equation (3) for each $ij \in E$. Solve the following SDP:

$$\max \sum_{ij \in E} w_{ij} \left(\frac{\text{Tr}[\mathcal{O}_{ij}]}{4} + \text{Tr}[C_{ij}M] \right) \quad (5)$$

$$\text{s.t.} \quad M(\mathbb{I}, \mathbb{I}) = 1, \quad (6)$$

$$M(\sigma_i^k, \sigma_i^k) = 1 \quad \forall i \in [n] \text{ and } k \in [3], \quad (7)$$

$$M(\sigma_i^k, \sigma_i^l) = 0 \quad \forall i \in [n] \text{ and } k \neq l \in [3], \quad (8)$$

$$M(\sigma_i^k, \sigma_j^l) = \text{Tr}[\sigma^k \otimes \sigma^l \rho_{ij}] \quad \forall ij \in E \text{ and } k, l \in [3], \quad (9)$$

$$M(\sigma_i^k, \mathbb{I}) = \text{Tr}[\sigma^k \otimes \mathbb{I} \rho_{ij}] \quad \forall ij \in E \text{ and } k \in [3], \quad (10)$$

$$M(\sigma_j^l, \mathbb{I}) = \text{Tr}[\mathbb{I} \otimes \sigma^l \rho_{ij}] \quad \forall ij \in E \text{ and } l \in [3], \quad (11)$$

$$\text{Tr}[\rho_{ij}] = 1 \quad \forall ij \in E, \quad (12)$$

$$\mathcal{H}(\mathbb{C}^{4 \times 4}) \ni \rho_{ij} \succeq 0 \quad \forall ij \in E, \quad (13)$$

$$\mathcal{S}(\mathbb{R}^{(3n+1) \times (3n+1)}) \ni M \succeq 0, \quad (14)$$

where $\mathcal{S}(\cdot)$ and $\mathcal{H}(\cdot)$ refer to the symmetric and Hermitian matrices, respectively.

► **Theorem 10.** The mathematical program of Problem 9 is an efficiently computable semidefinite program that provides an upper bound on $\lambda_{\max}(\sum_{ij \in E} H_{ij})$.

Proof. Constraints (6)–(12) are linear equalities on the entries of PSD matrices M and ρ_{ij} $\forall ij \in E$, hence we do indeed have an SDP. Since there are polynomially many variables of polynomial size, the usual considerations show computational efficiency, i.e. the program can be solved to arbitrary additive precision in polynomial time.

A larger matrix $X \succeq 0$, consisting of M and the ρ_{ij} as its diagonal blocks may be used to put the SDP into a more standard form (e.g. [7], Section 4.6.2). Although the ρ_{ij} are complex, the SDP may be solved as a real SDP by appealing to the standard approach of tracking the real and imaginary parts separately and observing $X \succeq 0$ if and only if

$$\begin{bmatrix} \text{Re}(X) & -\text{Im}(X) \\ \text{Im}(X) & \text{Re}(X) \end{bmatrix} \succeq 0.$$

Let $|\psi\rangle$ be an eigenvector corresponding to $\lambda_{\max}(\sum_{ij \in E} H_{ij})$, and let ρ_{ij}^* , $\forall ij \in E$, be the 2-qubit marginal density matrices of $\rho = |\psi\rangle\langle\psi|$, so that Constraints (12) and (13) are satisfied for the ρ_{ij}^* . In addition consider the moment matrix M for $|\psi\rangle$ with respect to \mathcal{M} , as described above. The matrix M satisfies Constraints (6), (7), (9)–(11), and (14) by the definition of a moment matrix, since

$$\langle\psi| \sigma_i^k \otimes \sigma_j^l \otimes \mathbb{I}_{[n]\setminus\{i,j\}} |\psi\rangle = \text{Tr}[\sigma^k \otimes \sigma^l \rho_{ij}^*], \text{ for } 0 \leq k, l \leq 3. \quad (15)$$

Constraint (8) is the only one that remains. Note that the real part of M , $M^* := \text{Re}(M) \succeq 0$ since $M \succeq 0$. By Equation (15), for any $j \in [n]$ and $k \neq l \in [3]$, $M(\sigma_i^k, \sigma_i^l) = \pm i \text{Tr}[\sigma^m \otimes \mathbb{I} \rho_{ij}^*]$, where $m \in [3] \setminus \{k, l\}$. The quantities in Equation (15) are real since tensor products of Pauli operators are Hermitian. This implies that $M(\sigma_i^k, \sigma_i^l)$ for $k \neq l \in [3]$ is imaginary and more generally that M^* and the ρ_{ij}^* satisfy all the constraints.

Consider the objective value for this solution, $\sum_{ij \in E} w_{ij} (\text{Tr}[\mathcal{O}_{ij}]/4 + \text{Tr}[C_{ij}M^*]) =$

$$\sum_{ij \in E} w_{ij} \text{Tr}[\mathcal{O}_{ij} \rho_{ij}^*] = \text{Tr} \left[\sum_{ij \in E} H_{ij} \rho \right] = \lambda_{\max} \left(\sum_{ij \in E} H_{ij} \right),$$

where the first equality follows from Equation (4). It follows that the optimal solution to Problem 9 has value at least that of the optimal solution of QLH. ◀

2.3 Rounding Approach and Formal Statement of Results

Overview. In classical SDP-based rounding schemes, one typically seeks to randomly “round” unit vectors $\mathbf{v}_i \in \mathbb{R}^d$ to scalars $z_i \in \{\pm 1\}$ so that the expected value of $z_i z_j$ approximates $\mathbf{v}_i \cdot \mathbf{v}_j$. The seminal hyperplane rounding scheme of Goemans and Williamson [20] achieves this by selecting a random unit vector $\mathbf{r} \in \mathbb{R}^d$ and setting $z_i = \mathbf{r} \cdot \mathbf{v}_i / |\mathbf{r} \cdot \mathbf{v}_i|$.

Rounding solutions from SDP relaxations for QLH to product states generalizes this approach. Recall that a product state has the form $|\psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle$ where each $|\psi_i\rangle \in \mathbb{C}^2$ is a local state on qubit i . We obtain a density matrix $\rho = |\psi\rangle \langle \psi| = |\psi_1\rangle \langle \psi_1| \otimes \dots \otimes |\psi_n\rangle \langle \psi_n|$, which is a tensor product of single-qubit density matrices $\rho_i := |\psi_i\rangle \langle \psi_i|$. Any such ρ_i may be expressed in the Pauli basis as $\rho_i = \frac{1}{2}(\mathbb{I} + \theta_{i1}\sigma^1 + \theta_{i2}\sigma^2 + \theta_{i3}\sigma^3)$, where $\theta_{ik} = \text{Tr}[\sigma^k \rho_i] \in \mathbb{R}$ and $\sum_{k \in [3]} \theta_{ik}^2 = 1$. In particular, product states with $\theta_{i1} = \theta_{i2} = 0$ and $\theta_{i3}^2 = 1$ correspond precisely to the classical setting (see Appendix E in [30] for an explicit connection between the two). Product states exhibit no entanglement, and we may specify θ_{ik} independently for each qubit i . However, instead of producing a single $z_i^2 = 1$ as in the classical case, we must produce a unit vector $\theta_i = [\theta_{i1}, \theta_{i2}, \theta_{i3}] \in \mathbb{R}^3$ for each $i \in [n]$. Briët, de Oliveira Filho, and Vallentin were the first to consider such generalizations of scalars to unit vectors, in the context of the Grothendieck problem [12, 13], and their analysis has fueled recent approximation algorithms for QLH [19, 21].

The classical Goemans-Williamson rounding scheme obtains the unit vectors \mathbf{v}_i from a Cholesky decomposition of a PSD matrix $V^T V = R \succeq 0$. Taking the \mathbf{v}_i to be columns of V yields $R_{i,j} = \mathbf{v}_i \cdot \mathbf{v}_j$. Recent approximation algorithms [11, 19, 21] for QLH have mimicked this approach, as do we. Let $M^* \succeq 0$ be an optimal solution to Problem 9 (the ρ_{ij}^* are not necessary to describe the rounding algorithm). We find a Cholesky decomposition $V^T V = M^*$, and let $\mathbf{v}_{ik} \in \mathbb{R}^d$ be the column of V associated with σ_i^k for $i \in [n], k \in [3]$; we may assume $d \leq 3n + 1$. In addition we let \mathbf{v}_0 be the column of V corresponding to \mathbb{I} . These are unit vectors as a consequence of Constraints (6) and (7).

We will employ the same rounding algorithm for both the general and strictly quadratic cases. While previous related works [11, 19, 21] have in some cases had to rely on more sophisticated rounding schemes because they have been amenable to analysis, we are able to shed light on what is arguably the most natural generalization of the Goemans-Williamson approach. We draw $\mathbf{r} \sim \mathcal{N}(0, \mathbb{I}_d)$, i.e. a multivariate distribution over d independent and standard Gaussian variables. For each qubit, we obtain the desired vector $\theta_i = [\theta_{i1}, \theta_{i2}, \theta_{i3}]$ as:

$$[\mathbf{v}_{i1} \cdot \mathbf{r}, \mathbf{v}_{i2} \cdot \mathbf{r}, \mathbf{v}_{i3} \cdot \mathbf{r}] / Q_i,$$

where $Q_i := \sqrt{(\mathbf{v}_{i1} \cdot \mathbf{r})^2 + (\mathbf{v}_{i2} \cdot \mathbf{r})^2 + (\mathbf{v}_{i3} \cdot \mathbf{r})^2}$ is a normalization.

The classical Max Cut problem corresponds to a strictly quadratic Hamiltonian (see Appendix E in [30] for justification); however, classical Max 2-SAT and more general Max 2-CSP have 1-local terms (i.e., linear terms in $\{\pm 1\}$ variables). In contrast, strictly quadratic instances of QLHP serve as a quantum generalization of Max 2-SAT and Max 2-CSP that have no 1-local terms. In order to obtain effective classical approximations in the presence of 1-local terms, an additional vector \mathbf{v}_0 is necessary, representing (scalar) identity. Generally, the vector \mathbf{v}_0 is used in conjunction with more sophisticated rounding schemes (e.g. [27]) to obtain positive expectation from the 1-local terms. For the quantum case, relatively simple approaches suffice to get good approximations [21]. Using the vector \mathbf{v}_0 is not necessary for the strictly quadratic case, and including it does not affect its approximation.

Rounding algorithm. The rounding approach described above produces single-qubit density matrices:

$$\rho_i = \frac{1}{2} \left(\mathbb{I} + \frac{\mathbf{v}_{i1} \cdot \mathbf{r}}{Q_i} \sigma^1 + \frac{\mathbf{v}_{i2} \cdot \mathbf{r}}{Q_i} \sigma^2 + \frac{\mathbf{v}_{i3} \cdot \mathbf{r}}{Q_i} \sigma^3 \right).$$

Hence, on any 1-local term, $\mathbb{E}[\text{Tr}[\sigma^k \rho_i]] = \mathbb{E}[\mathbf{v}_{ik} \cdot \mathbf{r}/Q_i] = 0$ since Q_i is an even function and $\mathbf{v}_{ik} \cdot \mathbf{r}$ is an odd function in each entry of \mathbf{r} . Thus, in order to get a nontrivial approximation on 1-local terms, we will use the vector \mathbf{v}_0 to globally flip the sign of the θ_i vectors of all qubits, i.e. $\mathbf{v}_{ik} \cdot \mathbf{r}/Q_i \rightarrow \text{sign}(\mathbf{v}_0 \cdot \mathbf{r})(\mathbf{v}_{ik} \cdot \mathbf{r}/Q_i)$. Since $\text{sign}(\mathbf{v}_0 \cdot \mathbf{r}) \in \{\pm 1\}$, for quadratic objective terms this factor will cancel out, but for 1-local terms we will gain additional objective from the correlation of $\mathbf{v}_0 \cdot \mathbf{r}$ and $\mathbf{v}_{ik} \cdot \mathbf{r}$. Formally, we can state the rounding algorithm, which applies to any instance of QLH, in Algorithm 1.

■ **Algorithm 1** Hyperplane rounding for 2-Local Hamiltonian.

1. Given some instance of Problem 1 formulate and solve the corresponding instance of Problem 9. Let M^* be the optimal moment matrix obtained from Problem 9.
2. Find the Cholesky decomposition of M^* , obtaining Cholesky vectors $\mathbf{v}_0 \in \mathbb{R}^d$ and $\{\mathbf{v}_{ik} \in \mathbb{R}^d\}$ such that $M^*(\sigma_i^k, \sigma_j^l) = \mathbf{v}_{ik} \cdot \mathbf{v}_{jl}$ and $M^*(\mathbb{I}, \sigma_i^k) = \mathbf{v}_0 \cdot \mathbf{v}_{ik}$ for $i, j \in [n]$ and $k, l \in [3]$.
3. Let \mathbf{r} be a random vector with $\mathbf{r} \sim \mathcal{N}(0, \mathbb{I}_d)$.
4. For each qubit i , set $Q_i = \sqrt{(\mathbf{v}_{i1} \cdot \mathbf{r})^2 + (\mathbf{v}_{i2} \cdot \mathbf{r})^2 + (\mathbf{v}_{i3} \cdot \mathbf{r})^2}$, and set $\theta_{ik} = \text{sign}(\mathbf{v}_0 \cdot \mathbf{r})(\mathbf{v}_{ik} \cdot \mathbf{r}/Q_i)$ for $k \in [3]$.
5. Output the (pure) state:

$$\rho = \bigotimes_{i=1}^n \frac{1}{2} (\mathbb{I} + \theta_{i1} \sigma^1 + \theta_{i2} \sigma^2 + \theta_{i3} \sigma^3).$$

We will give the following approximation guarantees for QLHP:

► **Theorem 11.** Fix $k \in \{1, 2, 3\}$. Suppose we are given an instance of QLHP (Problem 2), $\{H_e\}$ where $H_e = w_e P_e \otimes \mathbb{I}_{[n] \setminus \{e_1, e_2\}}$ for $w_e \geq 0$ and P_e a projector of rank at least k , for all $e \in E$. Let M^* be the optimal moment matrix for the corresponding SDP relaxation, Problem 9, and let ρ be the random output of Algorithm 1. Then,

$$\mathbb{E} \left[\text{Tr} \left[\sum_{e \in E} H_e \rho \right] \right] \geq \alpha(k) \left(\sum_{e \in E} w_e \left(\frac{\text{rank}(P_e)}{4} + \text{Tr}[C_e M^*] \right) \right) \geq \alpha(k) \lambda_{\max} \left(\sum_{e \in E} H_e \right),$$

where

$$\alpha(k) = \begin{cases} 2/\pi - 1/4 \approx 0.387 & \text{if } k = 1 \\ 16/(9\pi) \approx 0.565 & \text{if } k = 2 \\ 3/8 + 11/(9\pi) \approx 0.764 & \text{if } k = 3. \end{cases}$$

► **Theorem 12.** If, in addition to the assumptions of Theorem 11, the P_e are strictly quadratic projectors, then the random output of Algorithm 1 satisfies:

$$\mathbb{E} \left[\text{Tr} \left[\sum_{ij \in E} H_{ij} \rho \right] \right] \geq \alpha(k) \lambda_{\max} \left(\sum_{ij \in E} H_{ij} \right),$$

where

$$\alpha(k) = \begin{cases} 22/(15\pi) \approx 0.467 & \text{if } k = 1 \\ 1/3 + 24/(25\pi) \approx 0.639 & \text{if } k = 2 \\ 1/2 + 388/(405\pi) \approx 0.804 & \text{if } k = 3. \end{cases}$$

The above results are rigorous, but non-optimal. The quadratic analysis depends crucially on an expansion of a particular expectation in Hermite polynomials. One can consider a higher order Hermite series to more accurately capture the expectation and achieve a better approximation factor. We have such results, but opt to not include them in the interest of the reader. Higher orders bring increased tedium, and our technique should be clear enough at the end of the paper that an interested reader could do the higher order calculation.

One can ask, why not include a high enough order that the result becomes essentially optimal? The issue is that polynomial expansions often converge slowly in the presence of discontinuities [33]. Indeed, computationally we have determined that to get essentially optimal results one would need to go to high enough order that the polynomial expansion would become intractable. One can determine the optimal approximation factor by using a high order expansion and numerically optimizing or simply by randomly sampling over some “net” of the parameter space. Our observed approximation factors under these approaches are stated in Conjecture 6. Proving an approximation factor as large as the observed performance of our algorithm is the subject of future work.

2.4 Analysis Overview

We present an overview of our analysis for the strictly quadratic case, which will also carry over to the general case with additional bookkeeping and bounding for the 1-local terms. Suppose we are given an instance of QLHP (Problem 2) on which we execute Algorithm 1 to produce a random solution ρ . For $i, j \in [n]$, the 2-qubit marginals of ρ are

$$\rho_{ij} = \frac{1}{4}(\mathbb{I} + \theta_{i1}\sigma^1 + \theta_{i2}\sigma^2 + \theta_{i3}\sigma^3) \otimes (\mathbb{I} + \theta_{j1}\sigma^1 + \theta_{j2}\sigma^2 + \theta_{j3}\sigma^3), \quad (16)$$

and the objective value of ρ is $\sum_{ij \in E} \text{Tr}[H_{ij}\rho] = \sum_{ij \in E} w_{ij} \text{Tr}[P_{ij}\rho_{ij}]$, where $\text{Tr}[P_{ij}\rho_{ij}]$ is the unweighted contribution to the objective value from edge ij . Let M^* and ρ_{ij}^* for $ij \in E$ be the SDP solution obtained by Algorithm 1, and let $\text{SDP}_{ij} := \text{Tr}[P_{ij}\rho_{ij}^*]$ be the unweighted contribution to the SDP objective value from edge ij . The approximation ratio, which we seek to bound from below, is consequently:

$$\alpha = \mathbb{E} \left[\frac{\sum_{ij \in E} w_{ij} \text{Tr}[P_{ij}\rho_{ij}]}{\sum_{ij \in E} w_{ij} \text{SDP}_{ij}} \right] = \frac{\sum_{ij \in E} w_{ij} \mathbb{E}[\text{Tr}[P_{ij}\rho_{ij}]]}{\sum_{ij \in E} w_{ij} \text{SDP}_{ij}}.$$

Observe that $\text{Tr}[P_{ij}\rho_{ij}] \geq 0$ and $\text{SDP}_{ij} \geq 0$ since P_{ij} , ρ_{ij} , and ρ_{ij}^* are all PSD. Since all the terms in the denominator are nonnegative, it follows from an elementary argument that

$$\alpha \geq \frac{\sum_{ij \in E} w_{ij} \mathbb{E}[\text{Tr}[P_{ij}\rho_{ij}]]}{\sum_{ij \in E} w_{ij} \text{SDP}_{ij}} \geq \min_{ij \in E} \frac{\mathbb{E}[\text{Tr}[P_{ij}\rho_{ij}]]}{\text{SDP}_{ij}}.$$

Thus it suffices to bound the approximation ratio for the worst case occurring on a single edge.

Bounding a worst-case edge. We now focus our attention on a single edge $e = 12$ on qubits 1, 2. We collect the vectors \mathbf{v}_{ik} , obtained from a Cholesky decomposition of the SDP solution M^* , into matrices $V_i = [\mathbf{v}_{i1}, \mathbf{v}_{i2}, \mathbf{v}_{i3}] \in \mathbb{R}^{d \times 3}$, for $i = 1, 2$. We define an objective matrix $C \in \mathbb{R}^{3 \times 3}$, containing scaled 2-local Pauli-basis coefficients of P_{12} , in the vein of Equation (3): $C(\sigma_1^k, \sigma_2^l) := \text{Tr}[\sigma^k \otimes \sigma^l P_{12}]$, for $k, l \in [3]$ (note that C is not symmetric). With these definitions in hand, observe that $V_i^T V_i = \mathbb{I}_3$, by the SDP constraints (7) and (8), and that $4\text{Tr}[C_{12}M^*] = \text{Tr}[V_1 C V_2^T]$. The hyperplane rounding produces unit vectors $\theta_i^T = [\theta_{i1}, \theta_{i2}, \theta_{i3}] = \mathbf{r}^T V_i / \|V_i^T \mathbf{r}\|$. In terms of these variables, we have:

$$\begin{aligned} SDP_e &= \text{Tr}[P_{12}\rho_{12}^*] = \frac{1}{4} (\text{rank}(P_{12}) + \text{Tr}[V_1 C V_2^T]), \text{ and} \\ \mathbb{E}[APX_e] &= \mathbb{E}[\text{Tr}[P_{12}\rho_{12}]] = \frac{1}{4} (\text{rank}(P_{12}) + \mathbb{E}[\theta_1^T C \theta_2]) = \\ &= \frac{1}{4} \left(\text{rank}(P_{12}) + \mathbb{E}_{\mathbf{r}} \left[\frac{\mathbf{r}^T V_1 C V_2^T \mathbf{r}}{\|V_1^T \mathbf{r}\| \|V_2^T \mathbf{r}\|} \right] \right), \end{aligned}$$

by Equation (4) and because $\text{Tr}[P_{12}] = \text{rank}(P_{12})$, since P_{12} is a projector. Thus, setting $k = \text{rank}(P_{12})$, the quantity we seek to bound is

$$\alpha \geq \min_{V_1, V_2, C} \frac{k + \mathbb{E}_{\mathbf{r}} \left[\frac{\mathbf{r}^T V_1 C V_2^T \mathbf{r}}{\|V_1^T \mathbf{r}\| \|V_2^T \mathbf{r}\|} \right]}{k + \text{Tr}[V_1 C V_2^T]}.$$

The bulk of our analysis lies in (i) simplifying the above to reduce the number of parameters in the minimization and expectation (Appendix B in [30]), and (ii) deriving analytical bounds on the expectation (Appendix C in [30]).

Simplifying the Gaussian expectation. The first simplification follows from observing that $V_i^T \mathbf{r} \in \mathbb{R}^3$ are multivariate Gaussians for $i = 1, 2$ since they are linear combinations of Gaussians, $\mathbf{r} \sim \mathcal{N}(0, \mathbb{I})$. If we let $\mathbf{z}^T = [z_1, z_2, z_3] = \mathbf{r}^T V_1$ and $(\mathbf{z}')^T = [z'_1, z'_2, z'_3] = \mathbf{r}^T V_2$, then $[\mathbf{z}, \mathbf{z}'] \sim \mathcal{N}(0, \Sigma)$, where

$$\Sigma = \begin{bmatrix} \mathbb{I} & V_1^T V_2 \\ V_2^T V_1 & \mathbb{I} \end{bmatrix} \in \mathbb{R}^{6 \times 6}.$$

The Gaussians z_i are mutually independent as well as the z'_i , and the covariance between \mathbf{z} and \mathbf{z}' is given by $M = V_1^T V_2 \in \mathbb{R}^{3 \times 3}$. Our bound now depends on a constant number of parameters, the 18 entries of C and M :

$$\alpha \geq \min_{M, C} \frac{k + \mathbb{E}_{\mathbf{z}, \mathbf{z}'} \left[\frac{\mathbf{z}^T C \mathbf{z}'}{\|\mathbf{z}\| \|\mathbf{z}'\|} \right]}{k + \text{Tr}[C^T M]}. \quad (17)$$

For classical hyperplane rounding algorithms, C and M simply reduce to scalars, and one may resort to a numerical argument to furnish the desired bound. However, in the case of QLH above, numerical bounds exhibit poor precision or convergence due to the number of parameters. Thus we press on, and our next observation is that only the singular values of M matter for the analysis. The above arguments are detailed in Lemma 15 in [30], which also shows that we may assume:

$$C = \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & r \end{bmatrix} \text{ and } M = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}, \quad (18)$$

where a, b, c are the singular values of $V_1^T V_2$. This reduction to 6 parameters puts analysis of α within reach. The special case when $a = b = c$ turns out to be equivalent to the recently studied quantum analog of Max Cut related to the quantum Heisenberg model [2, 19]. For this case, a representation of the expectation,

$$\mathbb{E}_{\mathbf{z}, \mathbf{z}'} \left[\frac{\mathbf{z}^T C \mathbf{z}'}{\|\mathbf{z}\| \|\mathbf{z}'\|} \right] \quad (19)$$

as a hypergeometric function follows from work of Briët, de Oliveira Filho, and Vallentin (the expectation ends up being equivalent to the one in Lemma 2.1 from [13], when $\mathbf{u} \cdot \mathbf{v}$ in the lemma equals $a = b = c$). To the best of our knowledge, no elementary representation is known when a, b, c may be distinct. We appeal to Hermite analysis to express the expectation (19) as a polynomial that we are subsequently able to bound; this is carried out in Lemmas 17-18/Appendix C of [30].

Introducing constraints from positivity. The matrices C and M from (18) are related to the quadratic Pauli-basis coefficients of P_{12} and ρ_{12}^* , respectively. The other ingredient of our analysis of the bound (17) is restricting C and M based on the facts that $P_{12} \succeq 0$ and $\rho_{12}^* \succeq 0$, which is undertaken in Appendix B.1 and Lemma 14 of [30]. This is where the SDP constraint (13) is used. The bound we obtain is

$$\alpha \geq \min_{\substack{[a,b,c] \in \mathcal{S} \\ [p,q,r] \in \mathcal{P}_k}} \frac{k + \mathbb{E}_{\mathbf{z}, \mathbf{z}'} \left[\frac{pz_1 z'_1 + qz_2 z'_2 + rz_3 z'_3}{\sqrt{(z_1^2 + z_2^2 + z_3^2)((z'_1)^2 + (z'_2)^2 + (z'_3)^2)}} \right]}{k + ap + bq + cr}, \quad (20)$$

where \mathcal{S} and \mathcal{P}_k are specific polytopes (\mathcal{S} is a simplex as is \mathcal{P}_k for $k \neq 2$) derived from the positivity of P_{12} and ρ_{12}^* . We further observe in Lemma 34 in [30] that p, q, r may be fixed (e.g., for $k = 3$, we may take $p = q = r = 1$). Finally, Lemmas 17 and 18 in [30] derive the bounds in the main theorems 11 and 12, respectively.

Analysis Highlights. We utilize Hermite polynomials to evaluate the expectation given in Equation (20). As the natural set of polynomials orthogonal under the expectation of Gaussian variables, [26], we obtain an expression for the expectation in terms of a convergent series which we can then truncate and bound to get rigorous results. Obtaining the coefficients in this series requires some additional facts about Hermite polynomials, and some combinatorial identities.

3 Conclusion

In this work we have demonstrated several new approximation algorithms for interesting cases of the Max 2-Local Hamiltonian problem. As is the theme in many works [4, 8, 18], we have given evidence that the geometry of 2-Local interactions can drastically effect approximability for traceless Hamiltonians since we demonstrate the bipartite case has a constant factor approximation algorithm and the unconstrained case is known to have no constant factor algorithm [11]. In addition to this, we have given a novel approximation algorithm and analysis for 2-Local Hamiltonian with local terms that are also projectors. This is especially interesting given the scarcity of approximation algorithms for quantum problems. Indeed, the rank 3 case, has been open for some time [18, 21]. Furthermore, we have provided new techniques for rounding to product states that we believe will have additional applications

in quantum information. Our rounding algorithm is quite natural given the solution of the SDP, and the ability to understand the expectation through Hermite polynomial analysis seems likely to extend to other kinds of Hamiltonians or problems.

References



- 1 Dorit Aharonov, Itai Arad, and Thomas Vidick. The quantum pcg conjecture, 2013. [arXiv:1309.7495](#).
- 2 Anurag Anshu, David Gosset, and Karen Morenz. Beyond Product State Approximations for a Quantum Analogue of Max Cut. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2020.7.
- 3 Sanjeev Arora, Eli Berger, Hazan Elad, Guy Kindler, and Muli Safra. On non-approximability for quadratic programs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 206–215. IEEE, 2005.
- 4 Nikhil Bansal, Sergey Bravyi, and Barbara M. Terhal. Classical approximation schemes for the ground-state energy of quantum and classical ising spin hamiltonians on planar graphs. *Quantum Info. Comput.*, 9(7):701–720, 2009.
- 5 Salman Beigi and Peter W Shor. On the complexity of computing zero-error and holevo capacity of quantum channels. *arXiv preprint arXiv:0709.2090*, 2007.
- 6 Hans Bethe. Zur theorie der metalle. *Zeitschrift für Physik*, 71(3-4):205–226, 1931.
- 7 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi:10.1017/CB09780511804441.
- 8 Fernando GSL Brandao and Aram W Harrow. Product-state approximations to quantum states. *Communications in Mathematical Physics*, 342(1):47–80, 2016.
- 9 Mark Braverman, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. The grothendieck constant is strictly smaller than krivine’s bound. In *Forum of Mathematics, Pi*, volume 1. Cambridge University Press, 2013.
- 10 Sergey Bravyi. Efficient algorithm for a quantum analogue of 2-sat. *Contemporary Mathematics*, 536:33–48, 2011.
- 11 Sergey Bravyi, David Gosset, Robert König, and Kristan Temme. Approximation algorithms for quantum many-body problems. *Journal of Mathematical Physics*, 60(3):032203, 2019.
- 12 Jop Briët, Fernando Mário de Oliveira Filho, and Frank Vallentin. The positive semidefinite grothendieck problem with rank constraint. In *International Colloquium on Automata, Languages, and Programming*, pages 31–42. Springer, 2010.
- 13 Jop Briët, Fernando Mário de Oliveira Filho, and Frank Vallentin. Grothendieck inequalities for semidefinite programs with rank constraint. *Theory of Computing*, 10(4):77–105, 2014. doi:10.4086/toc.2014.v010a004.
- 14 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck’s inequality. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 54–60. IEEE, 2004.
- 15 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.
- 16 Andrew C Doherty, Yeong-Cherng Liang, Ben Toner, and Stephanie Wehner. The quantum moment problem and bounds on entangled multi-prover games. In *2008 23rd Annual IEEE Conference on Computational Complexity*, pages 199–210. IEEE, 2008.
- 17 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998. doi:10.1145/285055.285059.
- 18 Sevag Gharibian and Julia Kempe. Approximation algorithms for qma-complete problems. *SIAM Journal on Computing*, 41(4):1028–1050, 2012.

- 19 Sevag Gharibian and Ojas Parekh. Almost Optimal Classical Approximation Algorithms for a Quantum Generalization of Max-Cut. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.31.
- 20 Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.
- 21 Sean Hallgren, Eunou Lee, and Ojas Parekh. An approximation algorithm for the max-2-local hamiltonian problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.
- 22 Aram W Harrow and Ashley Montanaro. Extremal eigenvalues of local hamiltonians. *Quantum*, 1:6, 2017.
- 23 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.
- 24 Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, 2006.
- 25 Alexei Yu Kitaev, Alexander Shen, Mikhail N Vyalyi, and Mikhail N Vyalyi. *Classical and Quantum Computation*. Number 47. American Mathematical Soc., 2002.
- 26 Nikolai N. Lebedev (author) and Richard Silverman (translator). *Special Functions and their Applications*. Courier Corporation, 1972.
- 27 Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 67–82. Springer, 2002.
- 28 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi:10.1017/CB09780511976667.
- 29 Tobias J Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, 2012.
- 30 Ojas Parekh and Kevin Thompson. Beating random assignment for approximating quantum 2-local hamiltonian problems, 2020. arXiv:2012.12347.
- 31 Ojas Parekh and Kevin Thompson. Application of the Level-2 Quantum Lasserre Hierarchy in Quantum Approximation Algorithms. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.102.
- 32 Stephen Piddock and Ashley Montanaro. The complexity of antiferromagnetic interactions and 2d lattices. *arXiv preprint arXiv:1506.04014*, 2015.
- 33 Joanna Piotrowska, Jonah M Miller, and Erik Schnetter. Spectral methods in the presence of discontinuities. *Journal of Computational Physics*, 390:527–547, 2019.
- 34 Stefano Pironio, Miguel Navascués, and Antonio Acín. Convergent relaxations of polynomial optimization problems with noncommuting variables. *SIAM Journal on Optimization*, 20(5):2157–2180, 2010.
- 35 Norbert Schuch and Frank Verstraete. Computational complexity of interacting electrons and fundamental limitations of density functional theory. *Nature Physics*, 5(10):732–735, 2009.
- 36 Henrik Sjögren. *Rigorous Analysis of Approximation Algorithms for MAX 2-CSP*. Skolan för datavetenskap och kommunikation, Kungliga Tekniska högskolan, 2009.
- 37 Vijay V Vazirani. *Approximation Algorithms*. Springer Science & Business Media, 2013.
- 38 David P Williamson and David B Shmoys. *The Design of Approximation Algorithms*. Cambridge university press, 2011.

Additive Sparsification of CSPs

Eden Pelleg 

Mathematical Institute, University of Oxford, UK

Stanislav Živný  

Department of Computer Science, University of Oxford, UK

Abstract

Multiplicative cut sparsifiers, introduced by Benczúr and Karger [STOC'96], have proved extremely influential and found various applications. Precise characterisations were established for sparsifiability of graphs with other 2-variable predicates on Boolean domains by Filtser and Krauthgamer [SIDMA'17] and non-Boolean domains by Butti and Živný [SIDMA'20].

Bansal, Svensson and Trevisan [FOCS'19] introduced a weaker notion of sparsification termed “additive sparsification”, which does not require weights on the edges of the graph. In particular, Bansal et al. designed algorithms for additive sparsifiers for cuts in graphs and hypergraphs.

As our main result, we establish that *all* Boolean Constraint Satisfaction Problems (CSPs) admit an additive sparsifier; that is, for every Boolean predicate $P : \{0, 1\}^k \rightarrow \{0, 1\}$ of a fixed arity k , we show that $\text{CSP}(P)$ admits an additive sparsifier. Under our newly introduced notion of all-but-one sparsification for non-Boolean predicates, we show that $\text{CSP}(P)$ admits an additive sparsifier for *any* predicate $P : D^k \rightarrow \{0, 1\}$ of a fixed arity k on an arbitrary finite domain D .

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Sparsification and spanners

Keywords and phrases additive sparsification, graphs, hypergraphs, minimum cuts

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.75

Related Version *Full Version*: <https://arxiv.org/abs/2106.14757> [29]

Funding This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors’ views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein. *Stanislav Živný*: Royal Society University Research Fellowship.

1 Introduction

Graph sparsification is the problem of, given a graph $G = (V, E)$ with quadratically many (in $|V|$) edges, finding a sparse subgraph $G_\varepsilon = (V, E_\varepsilon \subseteq E)$ such that important properties of G are preserved in G_ε . Sparse in this context usually means with sub-quadratically many edges, though in this work we require (and can achieve) linearly many edges.

One of the most studied properties of preservation is the size of cuts. If $G = (V, E, w)$ is an undirected weighted graph with $w : E \rightarrow \mathbb{R}_{>0}$, given some $S \subseteq V$, the cut of S in G is

$$\text{Cut}_G(S) = \sum_{\substack{\{u,v\} \in E \\ |\{u,v\} \cap S| = 1}} w(\{u,v\}),$$

the sum of weights of all edges connecting S and $S^c = V \setminus S$. In an influential paper, Benczúr and Karger [11] introduced cut sparsification with a multiplicative error. In particular, [11] showed that for any graph $G = (V, E, w)$ and any error parameter $0 < \varepsilon < 1$, there exists a sparse subgraph $G_\varepsilon = (V, E_\varepsilon \subseteq E, w')$ with $O(n(\log n)\varepsilon^{-2})$ edges (and new weights w' on



© Eden Pelleg and Stanislav Živný;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 75; pp. 75:1–75:15

Leibniz International Proceedings in Informatics



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

the edges in E_ε), such that for every $S \subseteq V$ we have

$$\text{Cut}_{G_\varepsilon}(S) \in (1 \pm \varepsilon) \text{Cut}_G(S).$$

This was later improved by Batson, Spielman and Srivastava [10] to a subgraph with $O(n\varepsilon^{-2})$ many edges. Andoni, Chen, Krauthgamer, Qin, Woodruff and Zhang showed that the dependency on ε is optimal [4].

The ideas from cut sparsification paved the way to various generalisations, including streaming [1], sketching [4], cuts in hypergraphs [23, 27], spectral sparsification [33, 32, 34, 21, 31] and the consideration of other predicates besides cuts [20]. In this work, we focus on the latter.

The cut sparsification result in [10] was explored for other Boolean binary predicates by Filtser and Krauthgamer [20], following a suggestion to do so by Kogan and Krauthgamer in [23]. Filtser and Krauthgamer found [20] a necessary and sufficient condition on the predicate for the graph to be sparsifiable (in the sense of [10]). In particular, [20] showed that not all Boolean binary predicates are sparsifiable. Later, Butti and Živný [14] generalised the result from [20] to arbitrary finite domain binary predicates.

We remark that [20, 14] use the terminology of *constraint satisfaction problems* (CSPs) with a fixed predicate P . This is equivalent to a (hyper)graph G with a fixed predicate. Indeed, the vertices of G correspond to the variables of the CSP and the (hyper)edges of G correspond to the constraints of the CSP. If the fixed predicate P is not symmetric, the (hyper)edges of G are directed. We will mostly talk about sparsification of (hyper)graphs with a fixed predicate but this is equivalent to the CSP view.

Recently, while trying to eliminate the requirement for the introduction of new weights for the sparse subgraph, Bansal, Svensson and Trevisan [7] have come up with a new sparsification notion with an additive error term. They have shown (cf. Theorem 3 in Section 2) that under their notion any undirected unweighted hypergraph has a sparse subhypergraph which preserves all cuts up to some additive term.

Motivation

The relatively recent notion of additive sparsification has not yet been explored to the same extent as the notion of multiplicative sparsification has been. We believe that this notion has a lot of potential for applications as the sparsifiers are not weighted, unlike multiplicative sparsifiers. Indeed, the main restriction of multiplicative sparsifiers in applications appears to be the number of distinct weights required in sparsifiers. For some graphs (such as the “barbell graph” – two disjoint cliques joined by a single edge), any nontrivial multiplicative sparsifier requires edges of different weights. In any case, the authors find the notion of additive sparsification interesting in its own right, independently of applications.

The goal of our work is to understand how the notion of additive sparsification developed in [7] for cuts behaves on (hyper)graphs with other predicates (beyond cuts), deriving inspiration from the generalisations of cuts to other predicates in the multiplicative setting established in [20, 14]. In particular, already Boolean binary predicates include interesting predicates such as the *uncut edges* (using the predicate $P(x, y) = 1$ iff $x = y$), *covered edges* (using the predicate $P(x, y) = 1$ iff $x = 1$ or $y = 1$), or *directed cut edges* (using the predicate $P(x, y) = 1$ iff $x = 0$ and $y = 1$). While such graph problems are well-known and extensively studied, it is not clear whether one should expect them to be sparsifiable or not. For instance, as mentioned before, not all (even Boolean binary) predicates are sparsifiable multiplicatively [20]. Are there some predicates that are not additively sparsifiable?

1.1 Contributions

Boolean predicates

Our main result, Theorem 9 in Section 3, shows that *all* hypergraphs with *constant* uniformity k , directed or undirected, admit additive sparsification with respect to all Boolean predicates $P : \{0, 1\}^k \rightarrow \{0, 1\}$; the number of hyperedges of the sparsifier with error $\varepsilon > 0$ is $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$, where the $O(\cdot)$ hides a factor that depends on k . This result has three ingredients. First, we observe that the result in [7] also holds true for directed hypergraphs. Second, we use a reduction via the k -partite k -fold covers of hypergraphs to the already solved case of Boolean Cut. Finally, we use linear algebra to prove the correctness of the reduction. While the reduction via the k -partite k -fold cover was used in previous works on multiplicative sparsification [20, 14], the subsequent non-trivial linear-algebraic analysis (Proposition 13) is novel and constitutes our main technical contribution, as well as our result that, unlike in the multiplicative setting, all Boolean predicates can be (additively) sparsified. We also show that our results immediately apply to the more general setting where different hyperedges are associated with different predicates (cf. Remark 10). This corresponds to CSPs with a fixed constraint language (of a finite size) rather than just a single predicate.

Non-Boolean predicates

We introduce a notion of sparsification that generalises the Boolean case to predicates on non-Boolean domains, i.e. a notion capturing predicates of the form $P : D^k \rightarrow \{0, 1\}$, where D is an arbitrary fixed finite set with $|D| \geq 2$. We call this type of sparsification “all-but-one” sparsification since the additive error term includes the maximum volume of $|D| - 1$ (out of $|D|$) parts, where the volume of a subset is the sum of the degrees in the subset. (The precise definition can be found in Section 4.) By building on the techniques used to establish our main result, we show that all hypergraphs (again, directed or undirected) admit additive all-but-one sparsification with respect to all predicates. This is stated as Theorem 20 in Section 4. We also show, in Section 5, that our notion of all-but-one sparsification is, in some sense, optimal.

Comparison to previous work

As mentioned above, our sparsifiability result is obtained by a reduction via the k -partite k -fold cover to the cut case established in [7]. A reduction via the k -partite k -fold cover was also used (for $k = 2$) in previous work on multiplicative sparsification [20, 14]. In particular, the correctness of the reduction for Boolean binary predicates in [20] is done via an ad hoc case analysis for 11 concrete predicates. In the generalisation to binary predicates on arbitrary finite domains in [14], the correctness is proved via a combinatorial property of bipartite graphs without a certain 4-vertex graph¹ as a subgraph and a reduction to cuts with more than two parts.

In our case, we use the same black-box reduction via the k -partite k -fold cover. Thus the reduction itself is pretty straightforward, although the analysis is not. In fact, we find it surprising and unexpected that the k -partite k -fold cover works in the additive setting. Our key contribution is the proof of its correctness. A few simple reductions get us to the most technically involved case, in which k is even and the k -ary predicate satisfies $P(1, \dots, 1) = 0$.

¹ A bipartite graph on four vertices with each part of size two and precisely one edge between the two parts.

Additive sparsifiability of such predicates is established in Proposition 13. Unlike in the multiplicative setting, it is not clear how to do this in a straightforward way similar to [20, 14]. Instead, we associate with a given predicate P a vector v_P in an appropriate vector space, identify special vectors that can be shown additively sparsifiable directly, show that linear combinations preserve sparsifiability, and argue that v_P can be generated by the special vectors. The latter is the most technical part of the proof. While there are several natural ideas how to achieve this in a seemingly simpler way (such as arguing that the special vectors form a basis), we have not managed to produce a simpler or shorter proof.

The result in [7] also works for non-constant k . We emphasise that we deal with constant k , which is standard in the CSP literature in that the predicate (or a set of predicates) is fixed and not part of the input. For constant k , the representation of predicates is irrelevant (cf. Remark 18). Thus we do not keep track of (and have not tried to optimise) the precise dependency of the reduction on the predicate arity k (or the domain size $q = |D|$).

Related work

The already mentioned spectral sparsification [33] is a stronger notion than cut sparsification as it requires that not only cuts but also the Laplacian spectrum of a given graph should be (approximately) preserved [32, 34, 21, 31, 7].

Our focus in this article is on *edge sparsifiers* (of cuts and generalisations via local predicates). There are also vertex sparsifiers, in which one reduces the number of vertices. Vertex sparsifiers have been studied for cut sparsification (between special vertices called terminals) [22, 26, 25, 15] as well as for spectral sparsification [24].

Sparsification in general is about finding a sparse sub(hyper)graph while preserving important properties of interest. In addition to cut sparsifiers, another well studied concept is that of *spanners*. A spanner of a graph is a (sparse) subgraph that approximately preserves distances of shortest paths. Spanners have been studied in great detail both in the multiplicative [5, 28, 3, 17, 6, 9, 30] and additive [2, 18, 12, 8, 35, 16] setting. Emulators are a generalisation of spanners in which the sparse graph is not required to be a subgraph of the original graph. We refer the reader to a nice recent survey of Elkin and Neimain for more details [19]. Some of the proofs are deferred to the full version of this paper [29].

2 Preliminaries

For an integer k , we denote by $[k]$ the set $\{0, 1, \dots, k-1\}$. All graphs and hypergraphs² in this paper are unweighted.

For an assignment $a : V \rightarrow S$ from the set of vertices of a (hyper)graph to some set S containing 0, we denote by $Z_a = \{v \in V : a(v) = 0\}$ the set of vertices mapped to 0.

If $0 \leq i \leq r^k - 1$ is an integer, we denote by $\text{rep}_{r,k}(i)$ the representation of i in base r as a vector in \mathbb{R}^k , where the first coordinate stands for the most significant digit, and the last coordinate for the least significant digit. For the special case $r = 2$, we use the notation $\text{bin}_k(i)$ for the binary representation of i .

We denote by $v[j]$ the j -th coordinate of the vector v , counting from 0.

For an integer $0 \leq i \leq 2^k - 1$, we use $\text{zeros}_k(i) = \{\ell \in [k] : \text{bin}_k(i)[\ell] = 0\}$; for example $\text{zeros}_6(52) = \{2, 4, 5\}$, since $\text{bin}_6(52) = (1, 1, 0, 1, 0, 0)$.

We now define the value of an assignment on a hypergraph with a fixed predicate.

² We use the standard definition of hypergraphs, in which every hyperedge is an ordered tuple of vertices.

► **Definition 1.** Let $G = (V, E)$ be a directed k -uniform hypergraph and let $P : D^k \rightarrow \{0, 1\}$ be a k -ary predicate on a finite set D . Given an assignment $a : V \rightarrow D$ of G , the value of a is defined by $\text{Val}_{G,P}(a) = \sum_{(v_1, \dots, v_k) \in E} P(a(v_1), \dots, a(v_k))$. If G is undirected and P is order invariant,³ we define $\text{Val}_{G,P}(a) = \sum_{\{v_1, \dots, v_k\} \in E} P(a(v_1), \dots, a(v_k))$.⁴

The notion of additive sparsification was first introduced in [7] for cuts in graphs and hypergraphs. In order to define it, we will need the $\text{Cut} : \{0, 1\}^k \rightarrow \{0, 1\}$ predicate defined by $\text{Cut}(b_1, \dots, b_k) = 1 \iff \exists i, j, b_i \neq b_j$. Given a hypergraph $G = (V, E)$ and a set $U \subseteq V$, we denote by $\text{vol}_G(U)$ the *volume* of U , defined as the sum of the degrees in G of all vertices in U .

► **Definition 2.** Let $G = (V, E)$ be an undirected k -uniform hypergraph, and denote $|V| = n$. We say that G admits additive cut sparsification with error ε using $O(f(n, \varepsilon))$ hyperedges if there exists a subhypergraph $G_\varepsilon = (V, E_\varepsilon \subseteq E)$ with $|E_\varepsilon| = O(f(n, \varepsilon))$, called an additive sparsifier of G , such that for every assignment $a : V \rightarrow \{0, 1\}$ we have

$$\left| \frac{|E|}{|E_\varepsilon|} \text{Val}_{G_\varepsilon, \text{Cut}}(a) - \text{Val}_{G, \text{Cut}}(a) \right| \leq \varepsilon(d_G|Z_a| + \text{vol}_G(Z_a)), \quad (1)$$

where d_G is the average degree of G .

Note that (1) can also be written as

$$\frac{|E|}{|E_\varepsilon|} \text{Val}_{G_\varepsilon, \text{Cut}}(a) \in \text{Val}_{G, \text{Cut}}(a) \pm \varepsilon(d_G|Z_a| + \text{vol}_G(Z_a)),$$

which explains the use of the term “additive” for the error.

Bansal, Svensson and Trevisan [7] showed the following sparsification result:

► **Theorem 3** (Additive Cut Sparsification [7, Theorem 1.3]). Let $G = (V, E)$ be an undirected n -vertex k -uniform hypergraph, and $\varepsilon > 0$. Then G admits additive cut sparsification with error ε using $O\left(\frac{n}{k}\varepsilon^{-2} \log\left(\frac{k}{\varepsilon}\right)\right)$ hyperedges.

► **Remark 4.** We call a predicate P *symmetric* if it is order invariant (as in Definition 1). Since Theorem 3 deals with only *undirected* hypergraphs, it is not clear how to generalise it to non-symmetric predicates directly, since the value of such predicates on undirected hypergraphs is not defined. Therefore, our course of action will be first to prove it for the case of directed hypergraphs, and then generalise it to other predicates on directed hypergraphs. In fact, by doing this we also prove the result for undirected hypergraphs with symmetric predicates, since hyperedges can be given arbitrary directions without changing the average degree of G , or the volume in G , or the value of the predicate in any assignment.

► **Remark 5.** Throughout this paper we only discuss the existence of sparsifiers and do not mention the time complexity to find them. However, the (implicit) time complexity results from [7] apply in our more general setting as well since the sparsifiers we find are in fact the same sparsifiers for all predicates, including cuts (cf. Remark 17).

An important tool we use to prove our results is the k -partite k -fold cover of a hypergraph. This construction is a well known one, and has been used for multiplicative sparsification (for $k = 2$) in [20] and [14].

³ $P(b_1, \dots, b_k) = P(b_{\sigma(1)}, \dots, b_{\sigma(k)})$ for all $b_1, \dots, b_k \in D$ and every permutation σ on the set $\{1, \dots, k\}$.

⁴ The terms are well defined since P is order invariant.

► **Definition 6.** Let $G = (V, E)$ be a directed k -uniform hypergraph. The k -partite k -fold cover of G is the hypergraph $\gamma(G) = (V^\gamma, E^\gamma)$ where

$$V^\gamma = \{v^{(0)}, v^{(1)}, \dots, v^{(k-1)} : v \in V\},$$

$$E^\gamma = \{(v_1^{(0)}, v_2^{(1)}, \dots, v_k^{(k-1)}) : (v_1, \dots, v_k) \in E\}.$$

If G is undirected we define the cover in the same way except

$$E^\gamma = \{(v_1^{(0)}, v_2^{(1)}, \dots, v_k^{(k-1)}) : \{v_1, \dots, v_k\} \in E\},$$

so for each hyperedge in G we get $k!$ hyperedges in $\gamma(G)$ in this case.

If $k = 2$ then $\gamma(G)$ corresponds to the well-known *bipartite double cover* of G [13].

3 Sparsification of Boolean Predicates

As mentioned in Section 1, we begin by observing that Theorem 3 also works for directed hypergraphs. The simple proof of this fact can be found in the full version [29]. (We emphasise that we treat k as a constant, cf. Remark 18.)

► **Proposition 7.** Let $G = (V, E)$ be a directed n -vertex k -uniform hypergraph, and $\varepsilon > 0$. Then G admits additive cut sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.

From now on, whenever we say a “hypergraph”, we mean a “directed hypergraph” with n vertices. By Remark 4, the results also apply to undirected hypergraphs (whenever it makes sense, i.e. if the associated predicate is symmetric). We also omit the word additive when discussing sparsification. The following notion of sparsification is a natural generalisation of cut sparsification (Definition 2) to arbitrary predicates.

► **Definition 8.** Let P be a k -ary Boolean predicate and $G = (V, E)$ a k -uniform hypergraph. We say that G admits P -sparsification with error ε using $O(f(n, \varepsilon))$ hyperedges if there exists a subhypergraph $G_\varepsilon = (V, E_\varepsilon \subseteq E)$ with $|E_\varepsilon| = O(f(n, \varepsilon))$, called a P -sparsifier of G , such that for every assignment $a : V \rightarrow \{0, 1\}$ we have

$$\left| \frac{|E|}{|E_\varepsilon|} \text{Val}_{G_\varepsilon, P}(a) - \text{Val}_{G, P}(a) \right| \leq \varepsilon(d_G|Z_a| + \text{vol}_G(Z_a)), \quad (2)$$

where d_G is the average degree of G .

The following theorem is our main result, extending Proposition 7 to *all* k -ary predicates with Boolean domains.

► **Theorem 9 (Main).** For every k -uniform hypergraph G (k is a constant), every k -ary Boolean predicate $P : \{0, 1\}^k \rightarrow \{0, 1\}$, and every $\varepsilon > 0$, G admits P -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.

Theorem 9 can be informally restated as “every k -uniform hypergraph is sparsifiable with respect to all k -ary Boolean predicates” or “for every Boolean predicate P of constant arity, $\text{CSP}(P)$ is sparsifiable”.

► **Remark 10.** It is possible to consider an even more general case where each hyperedge in G has its own predicate. In this case, we can apply Theorem 9 to each of the hypergraphs obtained by taking only hyperedges corresponding to a specific predicate, and so get a sparsifier for each such predicate. Taking the union of all their hyperedges, we get a new

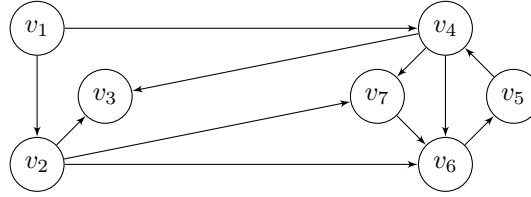
hypergraph G_ε , which is a sparsifier of the original hypergraph. Indeed, it has $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges since it is the union of a constant number of hypergraphs. (The number of predicates $P : \{0,1\}^k \rightarrow \{0,1\}$ is constant, since k is constant.) It also satisfies (2) for any given assignment up to some constant factor, since all the sparsifiers it is composed of do. This constant factor can be eliminated by choosing $\varepsilon_0 = \frac{\varepsilon}{m}$ for an appropriate m that depends only on k .

The main work in the proof of Theorem 9 is for even values of k ; a simple reduction (Proposition 16) then reduces the case of k odd to the even case.

In order to prove Theorem 9 for even k , we use the k -partite k -fold cover of G and apply Proposition 7 to various assignments of it. For a k -ary Boolean predicate $P : \{0,1\}^k \rightarrow \{0,1\}$, we consider the vector $v_P \in \mathbb{R}^{2^k}$, defined by $v_P[i] = P(\text{bin}_k(i))$. For instance, for the Cut predicate on a 3-uniform hypergraph, we have $v_{\text{Cut}} = (0, 1, 1, 1, 1, 1, 1, 0)$.

For a given hypergraph G and an assignment a , we consider the vector $v_{G,a} \in \mathbb{R}^{2^k}$ defined by $v_{G,a}[i] = |\{(v_1, \dots, v_k) \in E : (a(v_1), \dots, a(v_k)) = \text{bin}_k(i)\}|$. In other words, each coordinate of $v_{G,a}$ counts the hyperedges in G whose vertices are assigned some specific set of values by a .

► **Example 11.** Given the graph $G = (V, E)$ in Figure 1 (so $k = 2$) and the assignment $a : V \rightarrow \{0,1\}$ defined as $a(v_1) = a(v_2) = a(v_3) = 0$ and $a(v_4) = a(v_5) = a(v_6) = a(v_7) = 1$, we have $v_{G,a} = (2, 3, 1, 5)$, since there are two edges with assignment $(0,0)$, namely (v_1, v_2) and (v_2, v_3) , three edges with assignment $(0,1)$, namely (v_1, v_4) , (v_2, v_6) , and (v_2, v_7) , etc.



■ **Figure 1** Graph from Example 11.

Under these notations, we get $\text{Val}_{G,P}(a) = \langle v_P, v_{G,a} \rangle$, where $\langle \cdot, \cdot \rangle$ is the standard inner product in \mathbb{R}^{2^k} . We begin by proving the following useful lemma.

► **Lemma 12.** Let $G = (V, E)$ be a k -uniform hypergraph, P_1, \dots, P_m be k -ary Boolean predicates (m is a constant). Suppose that for every $\varepsilon > 0$ and $1 \leq i \leq m$, G admits P_i -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges, and that the same subhypergraph $G_\varepsilon = (V, E_\varepsilon \subseteq E)$ is a P_i -sparsifier for all P_i . Suppose that P is some k -ary Boolean predicate for which we have $v_P = \sum_{i=1}^m \lambda_i v_{P_i}$ for some constants $\lambda_1, \dots, \lambda_m \in \mathbb{R}$. Under these conditions, G admits P -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.

Proof. Let $\varepsilon > 0$ and denote $\varepsilon_i = \frac{\varepsilon}{m|\lambda_i|}$ (if $\lambda_i = 0$ take $\varepsilon_i = 1$ instead) and $\varepsilon_0 = \min\{\varepsilon_1, \dots, \varepsilon_m\}$. Let $G_{\varepsilon_0} = (V, E_{\varepsilon_0})$ be the common witness subhypergraph for ε_0 promised by the assumption. We know that every P_i satisfies

$$\left| \frac{|E|}{|E_{\varepsilon_0}|} \text{Val}_{G_{\varepsilon_0}, P_i}(a) - \text{Val}_{G, P_i}(a) \right| \leq \varepsilon_0 (d_G(Z_a) + \text{vol}_G(Z_a)) \quad (3)$$

for every assignment $a : V \rightarrow \{0,1\}$. We also have

$$\text{Val}_{G,P}(a) = \langle v_P, v_{G,a} \rangle = \left\langle \sum_{i=1}^m \lambda_i v_{P_i}, v_{G,a} \right\rangle = \sum_{i=1}^m \lambda_i \langle v_{P_i}, v_{G,a} \rangle = \sum_{i=1}^m \lambda_i \text{Val}_{G, P_i}(a),$$

75:8 Additive Sparsification of CSPs

and similarly

$$\text{Val}_{G_{\varepsilon_0}, P}(a) = \sum_{i=1}^m \lambda_i \text{Val}_{G_{\varepsilon_0}, P_i}(a).$$

Therefore, for every assignment a we get

$$\begin{aligned} \left| \frac{|E|}{|E_{\varepsilon_0}|} \text{Val}_{G_{\varepsilon_0}, P}(a) - \text{Val}_{G, P}(a) \right| &= \left| \frac{|E|}{|E_{\varepsilon_0}|} \sum_{i=1}^m \lambda_i \text{Val}_{G_{\varepsilon_0}, P_i}(a) - \sum_{i=1}^m \lambda_i \text{Val}_{G, P_i}(a) \right| \\ &\leq \sum_{i=1}^m |\lambda_i| \left| \frac{|E|}{|E_{\varepsilon_0}|} \text{Val}_{G_{\varepsilon_0}, P_i}(a) - \text{Val}_{G, P_i}(a) \right| \\ &\leq \sum_{i=1}^m |\lambda_i| \varepsilon_0 (d_G |Z_a| + \text{vol}_G(Z_a)) \\ &\leq \varepsilon (d_G |Z_a| + \text{vol}_G(Z_a)), \end{aligned}$$

where the second line is due to the triangle inequality, the third is due to (3) and the fourth is by the definition of ε_0 .

Furthermore, since m and all λ_i are constants,

$$|E_{\varepsilon_0}| = O\left(n \varepsilon_0^{-2} \log \frac{1}{\varepsilon_0}\right) = O\left(n \varepsilon^{-2} \log \frac{1}{\varepsilon}\right),$$

and so G_{ε_0} is a witness for the P -sparsification of G . ◀

The core of the proof of Theorem 9 is in the next proposition, which establishes the result for Boolean predicates on even uniformity hypergraphs, with a small restriction.

► **Proposition 13.** *Let k be an even number and G be a k -uniform hypergraph. Let P be a k -ary Boolean predicate with $P(1, 1, \dots, 1) = 0$. Then for every $\varepsilon > 0$, G admits P -sparsification with error ε using $O(n \varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.*

Proof. Let $\varepsilon > 0$. We consider $\gamma(G)$, the k -partite k -fold cover of G . Let $\gamma(G)_\varepsilon$ be a subhypergraph of $\gamma(G)$ promised by Proposition 7, and $G_\varepsilon = (V, E_\varepsilon)$ the corresponding subhypergraph of G , i.e. the subhypergraph which satisfies $\gamma(G_\varepsilon) = \gamma(G)_\varepsilon$ (by taking the hyperedges corresponding to the ones of $\gamma(G)_\varepsilon$).

Let $a : V \rightarrow \{0, 1\}$. For every subset $T \subseteq [k]$, we look at the assignment $a_T : V^\gamma \rightarrow \{0, 1\}$ defined by $a_T(v^{(i)}) = 0$ if $i \in T$ and $a(v) = 0$, and $a_T(v^{(i)}) = 1$ otherwise. We therefore have

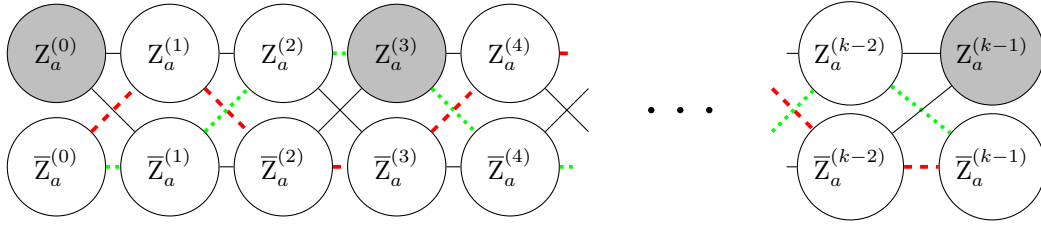
$$\left| \frac{|E^\gamma|}{|E_\varepsilon^\gamma|} \text{Val}_{\gamma(G)_\varepsilon, \text{Cut}}(a_T) - \text{Val}_{\gamma(G), \text{Cut}}(a_T) \right| \leq \varepsilon (d_{\gamma(G)} |Z_{a_T}| + \text{vol}_{\gamma(G)}(Z_{a_T})). \quad (4)$$

Define the vector $u_T \in \mathbb{R}^{2^k}$ as follows:

$$u_T[j] = \begin{cases} 1 & T \cap \text{zeros}(j) \neq \emptyset, [k] \\ 0 & \text{otherwise} \end{cases}.$$

In other words, the vector u_T is 1 in index j if and only if there exists an index $i \in T$ in which the binary representation of j has a zero, with the exception of $u_{[k]}[0] = 0$. Denote by P_T the predicate corresponding to u_T , that is $P_T(\text{bin}_k(j)) = 1 \iff u_T[j] = 1$. Observe that

$$\text{Val}_{\gamma(G), \text{Cut}}(a_T) = \text{Val}_{G, P_T}(a),$$



■ **Figure 2** An example of a representation of an assignment on $\gamma(G)$. $Z_a^{(i)}$ consists of all vertices in $V^{(i)}$ which are a copy of a vertex $v \in V$ with $a(v) = 0$, and $\bar{Z}_a^{(i)}$ consists of the rest of $V^{(i)}$. Each hyperedge has a unique path from left to right (but a path might belong to multiple hyperedges), choosing one of $Z_a^{(i)}, \bar{Z}_a^{(i)}$ for each i . Each such path is also in 1-1 correspondence with a coordinate in u_T . In this example $T = \{0, 3, k-1\}$ and the shaded sets represent $a_T^{-1}(0)$. By green dotted lines we indicated a path corresponding to a hyperedge counted in $\text{Val}_{\gamma(G), \text{Cut}}(a_T)$, and by red dashed lines we indicated a path which does not. The green dotted path corresponds to a value of 1 in the coordinate of u_T with binary representation $(1, 1, 0, 0, 1, \dots, 0, 1)$, and the red dashed path to a value 0 in the coordinate with binary representation $(1, 0, 1, 1, 0, \dots, 1, 1)$. Note that if $T = [k]$ then any hyperedge corresponding to a path only on $Z_a^{(i)}$ is not counted.

since they both count exactly hyperedges (v_1, \dots, v_k) which have some vertex v_i with $a(v_i) = 0$ with $i \in T$, but if $T = [k]$ then they do not count hyperedges which have $a(v_i) = 0$ for all $i = 1, \dots, k$ (see example in Figure 2). The same is true for any hypergraph, and in particular for G_ε , that is

$$\text{Val}_{\gamma(G_\varepsilon), \text{Cut}}(a_T) = \text{Val}_{G_\varepsilon, P_T}(a).$$

Putting these results in (4), we get

$$\begin{aligned} \left| \frac{|E|}{|E_\varepsilon|} \text{Val}_{G_\varepsilon, P_T}(a) - \text{Val}_{G, P_T}(a) \right| &\leq \varepsilon(d_{\gamma(G)}|Z_{a_T}| + \text{vol}_{\gamma(G)}(Z_{a_T})) \\ &\leq \varepsilon(d_G|Z_a| + \text{vol}_G(Z_a)), \end{aligned}$$

so G admits P_T sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges for every $T \subseteq [k]$, and for every ε the sparsification is witnessed by the same subhypergraph G_ε . (Notice that Proposition 7, when applied to $\gamma(G)$ which has kn vertices, gives us a subhypergraph with $O(kn\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges, and recall that k is a constant.)

Our next goal is to show that the vector v_P is a linear combination of the vectors u_T for all $T \in [k]$. To show that, we show that every vector e_r in the standard basis of \mathbb{R}^{2^k} , with $r \neq 2^k - 1$, is a linear combination of these vectors. This is sufficient since the last coordinate of v_P is 0 by the assumption. First we need to order the various sets T . We order them in the following decreasing lexicographic order $T_0, T_1, \dots, T_{2^k-1}$, where $T_j = \text{zeros}(j)$, so $T_0 = [k], T_1 = [k] \setminus \{k-1\}, T_2 = [k] \setminus \{k-2\}, T_3 = [k] \setminus \{k-1, k-2\}, T_4 = [k] \setminus \{k-3\}$ and so on, until $T_{2^k-1} = \emptyset$.

Let e_r be a vector in the standard basis of \mathbb{R}^{2^k} . We introduce the following coefficients for $0 \leq m \leq 2^k - 1$:

$$\lambda_{r,m} = \frac{1}{2}(-1)^{\text{Ham}(r \oplus m) + (1 - \mathbb{1}_{r \& m})},$$

where $\oplus, \&$ are the Xor and And binary functions respectively,⁵ Ham is the Hamming weight

⁵ The Xor of two integers is defined as the bitwise Boolean Xor of their binary representations, where the

75:10 Additive Sparsification of CSPs

function, and $\mathbf{1}_d$ returns 1 if $d \neq 0$ and 0 if $d = 0$. Denote

$$f_1(m) = \text{Ham}(r \oplus m) \quad , \quad f_2(m) = (1 - \mathbf{1}_{r \& m}).$$

We shall prove that

$$e_r = \sum_{m=0}^{2^k-1} \lambda_{r,m} u_{T_m}. \quad (5)$$

▷ **Claim 14.** The sum of all coefficients is 0; i.e., $\sum_{m=0}^{2^k-1} \lambda_{r,m} = 0$.

Proof. Let $b_1 b_2 \dots b_k$ be the binary representation of r . Since $r < 2^k - 1$, there exists some $1 \leq i \leq k$ for which $b_i = 0$. We can partition the coefficients into pairs, such that $\lambda_{r,m_1}, \lambda_{r,m_2}$ is a pair if and only if m_1, m_2 differ in the i -th coordinate only. This is clearly a partition. For each pair, f_1 gives m_1, m_2 different parity values, and f_2 gives them the same value (since $b_i = 0$), so $\lambda_{r,m_1}, \lambda_{r,m_2}$ have opposite signs, so their sum is zero. This is true for every pair, so the overall sum is zero, and the claim is proved. ◁

We prove (5) coordinate-wise. First we look at the coordinate r . Consider the set W of all vectors u_{T_m} for which the coordinate r is 0. If we show that the sum of the corresponding coefficients of the vectors in W is -1 , using Claim 14 we will deduce the result in this case. We distinguish 2 cases:

Case (I): $r = 0$. By the definition of u_T , in this case the set W contains two vectors, $u_{[k]}$ and u_\emptyset . The corresponding coefficients are $\lambda_{r,0} = -\frac{1}{2}$ and $\lambda_{r,2^k-1} = -\frac{1}{2}$ (since k is even), which sum up to -1 .

Case (II): $r > 0$. As in the proof of Claim 14, let $b_1 b_2 \dots b_k$ be the binary representation of r , and choose a coordinate $1 \leq i \leq k$ for which $b_i = 1$. Partition the vectors in W into pairs where $u_{T_{m_1}}, u_{T_{m_2}}$ is a pair if and only if m_1, m_2 differ in the i -th coordinate only. This is clearly a partition of all vectors, and by the definition of u_T , each such pair is either contained in W or disjoint from W so this is indeed a partition of W . (Note that $u_{T_{m_1}}[r]$ is determined by $T_{m_1} \cap \text{zeros}(r)$ which is in fact $\text{zeros}(m_1) \cap \text{zeros}(r)$, and the same for m_2 . Since m_1, m_2 differ in the i -th coordinate only, and r is not zero in this coordinate, this coordinate contributes nothing to the intersections, and so both these intersections are empty or non-empty together. The intersection never equals $[k]$ since $r > 0$.) For every such pair in W , if it does not contain the negation of $\text{bin}(r)$, then there is some other index $j \neq i$ in which r, m_1, m_2 are all 1. (This is because in all other coordinates m_1, m_2 are equal, and since they are not the negation of r , there is some coordinate $j \neq i$ in which they are equal to the j -th coordinate of r . These coordinates cannot be all 0, since this would imply $u_{T_{m_1}}, u_{T_{m_2}} \notin W$.) This implies that f_2 gives m_1, m_2 the same value, and clearly f_1 gives them different parity values, so $\lambda_{r,m_1} + \lambda_{r,m_2} = 0$. However, for the pair which contains the negation of r (this pair is clearly in W), suppose without loss of generality the negation is m_1 . Then f_2 gives m_1, m_2 the values 1, 0 respectively, and f_1 gives m_1 an even value and m_2 an odd value (since k is even), and so $\lambda_{r,m_1} = \lambda_{r,m_2} = -\frac{1}{2}$, and the overall sum is -1 . This finishes the proof of (5) in the coordinate r .

Now let $r' \neq r$ be some other coordinate, and let $c_1 c_2 \dots c_k$ be its binary representation. First, if $r' = 2^k - 1$ then for all m we have $u_{T_m}[r'] = 0$ by definition, so the linear combination

Boolean Xor of two bits is their sum modulo 2. The And of two integers is defined the same way with the Boolean And function which is defined as $\text{And}(i, j) = 1 \iff i = j = 1$.

of this coordinate is 0. So suppose $r' < 2^k - 1$. As before let W be the set of all vectors u_{T_m} for which the coordinate r' is 0. We show that the sum of the corresponding coefficients is zero, and again deduce the result using Claim 14. Now, there exists some index i for which $b_i \neq c_i$. Again we have two cases:

Case (1): $b_i = 0, c_i = 1$. Partition the vectors in W into pairs where $u_{T_{m_1}}, u_{T_{m_2}}$ is a pair if and only if m_1, m_2 differ in the i -th coordinate only. This is clearly a partition of all the vectors, and by the definition of u_T , each such pair is either contained in W or disjoint from W , so this is indeed a partition of W . For every such pair in W , f_1 gives m_1, m_2 different parity values, and f_2 gives them the same value (since $b_i = 0$), so $\lambda_{r, m_1}, \lambda_{r, m_2}$ have opposite signs, so their sum is zero. This is true for every pair in W , so the overall sum is zero.

Case (2) $b_i = 1, c_i = 0$. Here we consider two sub-cases:

Case (2a): $r' = 0$. The only vectors in W in this case are $u_{[k]}$ and u_\emptyset . The corresponding coefficients are $\lambda_{r, 0} = \frac{1}{2}(-1)^{\text{Ham}(r)+1}$ and $\lambda_{r, 2^k-1} = \frac{1}{2}(-1)^{\text{Ham}(\neg r)}$, where $\neg r$ denotes the negation of the binary representation of r . Since k is even, we know that $r, \neg r$ have the same parity, and so the sum of the two coefficients is 0.

Case (2b): $r' \neq 0$. Choose some j for which $c_j = 1$. Partition the vectors in W into pairs where $u_{T_{m_1}}, u_{T_{m_2}}$ is a pair if and only if m_1, m_2 differ in the j -th coordinate only. The argument for this being a partition of W is similar to the argument in Case (1). For each pair in W , f_1 gives m_1, m_2 a different parity as always, and f_2 gives them the same value, since r, m_1, m_2 are all 1 in the index i (similar argument as before), so the sum of coefficients is 0 for each pair, and so for all coefficients corresponding to vectors in W .

This finishes the proof of (5), and so v_P is a linear combination of the vectors u_T . From the result above and Lemma 12 we deduce that G admits P -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges, as required. ◀

To complete the picture for even k , we reduce to Proposition 13 by a simple “complementarity trick”; the proof can be found in the full version [29].

► **Proposition 15.** *Let k be an even number, and G a k -uniform hypergraph. Let P be a k -ary Boolean predicate. Then for every $\varepsilon > 0$, G admits P -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.*

The final piece in the jigsaw, proved in the full version [29], shows how to reduce sparsification of k -uniform hypergraphs with k odd to the case of $(k+1)$ -uniform hypergraphs by adding a universal vertex and extending the original predicate by one dimension.

► **Proposition 16.** *Let k be an odd number, and $G = (V, E)$ a k -uniform hypergraph. Let P be a k -ary Boolean predicate. Then for every $\varepsilon > 0$, G admits P -sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.*

Propositions 15 and 16 complete the proof of Theorem 9.

► **Remark 17.** In the proof of Proposition 13 the hypergraph G_ε was chosen independently of the predicate P . Since Propositions 15 and 16 reduce to that case, we have in fact shown that for every $\varepsilon > 0$, Theorem 9 is witnessed by the same subhypergraph G_ε for all different predicates P . This will be important in the proof of Theorem 20.

► **Remark 18.** We note that our main result, Theorem 9, extends Theorem 3 in the regime where k is a constant, which is the main focus of this paper. However, Theorem 3 also works for non-constant k [7]. If k is not a constant, it can be seen from the proof of Lemma 12 that the number of hyperedges of the sparse subhypergraph is multiplied by a factor of $O(m^2)$ (since $O(m)$ is the proportion between ε and ε_0 given that the coefficients λ_i are constant).

In Proposition 13 we have $m = 2^k$, and so for k not constant we get an additional factor of 4^k . Furthermore, in Propositions 7 and 13 we obtain extra factors of k , by considering the k -partite k -fold cover. While the regime with non-constant k is interesting for cuts, for arbitrary predicates one needs to be careful about representation as the natural (explicit) representation of (non-symmetric) predicates requires exponential space in the arity k .

4 Sparsification of Non-Boolean Predicates

We now focus on non-Boolean predicates; i.e., predicates of the form $P : D^k \rightarrow \{0, 1\}$ with $|D| > 2$. Without loss of generality, we assume $D = [q]$ for some $q \geq 2$. The most natural way of generalising Theorem 9 to larger domains appears to be to use the same bound with $Z_a = \{v \in V : a(v) = 0\}$. This, however, cannot give the desired sparsification result (cf. Section 5). Instead we use a different and somewhat weaker kind of generalisation of the Boolean case, and show that all hypergraphs are still sparsifiable with respect to all predicates using this definition.

► **Definition 19.** Let $P : D^k \rightarrow \{0, 1\}$ be a k -ary predicate where $D = [q]$. We say that a k -uniform hypergraph $G = (V, E)$ admits all-but-one P -sparsification with error ε using $O(f(n, \varepsilon))$ hyperedges if there exists a subhypergraph $G_\varepsilon = (V, E_\varepsilon \subseteq E)$ with $|E_\varepsilon| = O(f(n, \varepsilon))$ such that for every assignment $a : V \rightarrow D$ we have

$$\left| \frac{|E|}{|E_\varepsilon|} \text{Val}_{G_\varepsilon, P}(a) - \text{Val}_{G, P}(a) \right| \leq \varepsilon(d_G |M_a| + \text{vol}_G(N_a)), \quad (6)$$

where M_a is the largest set among the sets $\{v \in V : a(v) = i\}$, N_a is the set with the largest volume among the sets $\{v \in V : a(v) = i\}$ for $0 \leq i \leq q-2$, and d_G is the average degree in G .

Observe that the maximum in Definition 19 is over $i = 0, \dots, q-2$ without $i = q-1$, hence the name “all-but-one”. We note that there is nothing special about $q-1$ and any value from $[q]$ could be chosen in Definition 19.

Under Definition 19, Theorem 9 generalises (proof of can be found in the full version [29]).

► **Theorem 20.** For every k -uniform hypergraph $G = (V, E)$, every k -ary predicate $P : D^k \rightarrow \{0, 1\}$ with $D = [q]$ (k, q are constants), and every $\varepsilon > 0$, G admits P all-but-one sparsification with error ε using $O(n\varepsilon^{-2} \log \frac{1}{\varepsilon})$ hyperedges.

Note that in the case of $q = 2$ we have $P : \{0, 1\}^k \rightarrow \{0, 1\}$, and Definition 19 and Theorem 20 coincide with Definition 8 and Theorem 9. This is because when $q = 2$ the definitions of M_a, N_a coincide with the definition of Z_a in the Boolean case.

5 Optimality of All-But-One Sparsification

One might wonder if there is a different, perhaps stronger way to define sparsification for predicates on non-Boolean domains. The following example shows that all-but-one sparsification is optimal.

For a hypergraph $G = (V, E)$ and a fixed assignment $a : V \rightarrow [q]$ denote $S_i = \{v \in V : a(v) = i\}$ (so $S_0 = Z_a$). The definition of all-but-one sparsification lets us take a bound which depends on the sizes and volumes of all the sets S_i except for S_{q-1} . In fact, if we try to take a bound which depends on fewer of these sets, the definition fails to generalise even the most basic case of the Cut predicate. To see this, it is sufficient to consider the graph case,

i.e. $k = 2$. Let us suppose, without loss of generality, that our bound does not depend on S_{q-2}, S_{q-1} . Consider the predicate $\text{Cut} : [q]^2 \rightarrow \{0, 1\}$ defined by $\text{Cut}(x, y) = 1 \iff x \neq y$. A simple (but lengthy) argument in the full version [29] shows that cliques do not have a Cut-sparsifier using such a definition. In fact, the same argument works for any predicate P with $P(q-2, q-1) = P(q-1, q-2) = 1$ and $P(q-2, q-2) = P(q-1, q-1) = 0$. Thus if a definition does not depend on more than just S_{q-2}, S_{q-1} , it specifically does not depend on these two, so the same argument still works. Therefore, no definition with a bound which depends on “less” is possible, under the current assumptions.

References

- 1 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09), Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2009. doi:10.1007/978-3-642-02930-1_27.
- 2 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.
- 3 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.
- 4 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 7th ACM Conference on Innovations in Theoretical Computer Science (ITCS'16)*, pages 311–319. ACM, 2016. doi:10.1145/2840728.2840753.
- 5 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.
- 6 Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28(1):254–262, 1998. doi:10.1137/S009753979528826X.
- 7 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS'19)*, pages 910–928, 2019. doi:10.1109/FOCS.2019.00059.
- 8 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β) -spanners and purely additive spanners. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 672–681. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070526>.
- 9 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.
- 10 Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014. doi:10.1137/130949117.
- 11 András A. Benczúr and David R. Karger. Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ Time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96)*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.
- 12 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discret. Math.*, 19(4):1029–1055, 2005. doi:10.1137/S0895480103431046.
- 13 Richard A. Brualdi, Frank Harary, and Zevi Miller. Bigraphs versus digraphs via matrices. *Journal of Graph Theory*, 4(1):51–73, 1980. doi:10.1002/jgt.3190040107.
- 14 Silvia Butti and Stanislav Živný. Sparsification of binary CSPs. *SIAM J. Discret. Math.*, 34(1):825–842, 2020. doi:10.1137/19M1242446.

- 15 Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, Yunbum Kook, Yang P Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*, 2021. URL: <https://arxiv.org/abs/2007.07862>.
- 16 Shiri Chechik. New additive spanners. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 498–512. SIAM, 2013. doi:10.1137/1.9781611973105.36.
- 17 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998. doi:10.1137/S0097539794261295.
- 18 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000. doi:10.1137/S0097539797327908.
- 19 Michael Elkin and Ofer Neiman. Near-additive spanners and near-exact hopsets, A unified view. *Bull. EATCS*, 130, 2020. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/view/608/624>.
- 20 Arnold Filtser and Robert Krauthgamer. Sparsification of two-variable valued constraint satisfaction problems. *SIAM J. Discret. Math.*, 31(2):1263–1276, 2017. doi:10.1137/15M1046186.
- 21 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019. doi:10.1137/16M1091666.
- 22 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998. doi:10.1006/jcss.1998.1592.
- 23 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 6th ACM Conference on Innovations in Theoretical Computer Science (ITCS'15)*, pages 367–376. ACM, 2015. doi:10.1145/2688073.2688093.
- 24 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC'16)*, pages 842–850. ACM, 2016. doi:10.1145/2897518.2897640.
- 25 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'2010)*, pages 47–56. ACM, 2010. doi:10.1145/1806689.1806698.
- 26 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*, pages 3–12. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.28.
- 27 Ilan Newman and Yuri Rabinovich. On multiplicative lambda-approximations and some geometric applications. *SIAM J. Comput.*, 42(3):855–883, 2013. doi:10.1137/100801809.
- 28 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.
- 29 Eden Pelleg and Stanislav Živný. Additive Sparsification of CSPs, 2021. [arXiv:2106.14757](https://arxiv.org/abs/2106.14757).
- 30 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005. doi:10.1007/11523468_22.
- 31 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 2570–2581, 2019. doi:10.1137/1.9781611975482.159.
- 32 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011. doi:10.1137/080734029.
- 33 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM*

- Symposium on Theory of Computing (STOC'04)*, pages 81–90. ACM, 2004. doi:10.1145/1007352.1007372.
- 34 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. doi:10.1137/08074489X.
- 35 David P. Woodruff. Additive spanners in nearly quadratic time. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 6198 of *Lecture Notes in Computer Science*, pages 463–474. Springer, 2010. doi:10.1007/978-3-642-14165-2_40.

Faster Deterministic Modular Subset Sum

Krzysztof Potępa ✉

Jagiellonian University, Kraków, Poland

Abstract

We consider the *Modular Subset Sum* problem: given a multiset X of integers from \mathbb{Z}_m and a target integer t , decide if there exists a subset of X with a sum equal to $t \pmod{m}$. Recent independent works by Cardinal and Iacono (SOSA'21), and Axiotis et al. (SOSA'21) provided simple and near-linear algorithms for this problem. Cardinal and Iacono gave a randomized algorithm that runs in $\mathcal{O}(m \log m)$ time, while Axiotis et al. gave a deterministic algorithm that runs in $\mathcal{O}(m \text{ polylog } m)$ time. Both results work by reduction to a text problem, which is solved using a dynamic strings data structure.

In this work, we develop a simple data structure, designed specifically to handle the text problem that arises in the algorithms for Modular Subset Sum. Our data structure, which we call the *shift-tree*, is a simple variant of a segment tree. We provide both a hashing-based and a deterministic variant of the *shift-trees*.

We then apply our data structure to the Modular Subset Sum problem and obtain two algorithms. The first algorithm is Monte-Carlo randomized and matches the $\mathcal{O}(m \log m)$ runtime of the Las-Vegas algorithm by Cardinal and Iacono. The second algorithm is fully deterministic and runs in $\mathcal{O}(m \log m \cdot \alpha(m))$ time, where α is the inverse Ackermann function.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis; Theory of computation \rightarrow Algorithm design techniques

Keywords and phrases Modular Subset Sum, String Problem, Segment Tree, Data Structure

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.76

Acknowledgements I would like to thank Lech Duraj, Krzysztof Pióro, and Adam Polak for their help with preparing the paper, and the anonymous reviewers for their useful suggestions.

1 Introduction

The *Subset Sum* is a fundamental problem in computer science. It is defined as follows: given a multiset X of n positive integers and a target integer t , decide if there exists a subset of X , such that the sum of its elements is exactly t . The problem is known to be NP-complete [14], but only in a weak sense: a classic dynamic programming approach of Bellman [5] solves it in pseudo-polynomial $\mathcal{O}(nt)$ time. In recent years, there has been a lot of research towards improving the runtime [16, 17, 7, 12], which culminated in near-linear $\tilde{\mathcal{O}}(n + t)$ algorithms [7, 12].¹

In this work, we focus on the *Modular Subset Sum* problem. The *Modular Subset Sum* is a natural variant of the Subset Sum problem, where all sums are taken modulo m , for some given modulus m . We assume that the input multiset X is provided in a compact form: as a list of $\mathcal{O}(m)$ distinct elements along with their multiplicities. This assumption allows us to omit dependence on the number of elements n in algorithm complexities. Moreover, we focus on algorithms that return all possible subset sums, i.e. a set of all attainable values of t .

The dynamic programming of Bellman [5] can be easily adapted to solve the modular case in $\mathcal{O}(nm)$ time. Let S_i be the set of all attainable subset sums using only the first i elements. Bellman's algorithm iteratively computes the sets S_1, \dots, S_n using formula

¹ By writing $\tilde{\mathcal{O}}(f(n))$, we mean $\mathcal{O}(f(n) \text{ polylog } f(n))$.



$S_i = S_{i-1} \cup (S_{i-1} + x_i)$, where x_i is the i -th input element and $S_{i-1} + x_i = \{a + x_i : a \in S_{i-1}\}$. Most of the currently known improved algorithms simply simulate the consecutive iterations of Bellman's algorithm faster. An early notable exception is the $\tilde{O}(m^{5/4})$ algorithm of Koiliaris and Xu [16], which uses a divide-and-conquer approach based on results from number theory.

Abboud et al. [1] obtained a SETH-based conditional lower bound for the Subset Sum problem, which in particular implies that the Modular Subset Sum cannot be solved in $\mathcal{O}(m^{1-\varepsilon})$ time for any $\varepsilon > 0$. The first randomized algorithm that matched their lower-bound (up to subpolynomial factors) was introduced by Axiotis et al. in [4]. They achieved a running time of $\mathcal{O}(m \log^7 m)$ by simulating Bellman's dynamic programming faster using ideas from linear sketching.

Recently, simple and practical algorithms were provided independently in [9, 3]. Both results work by reducing the problem of computing Bellman's iteration to a text problem, but use different data structures to solve it efficiently. A Las-Vegas randomized $\mathcal{O}(m \log m)$ algorithm by Cardinal and Iacono [9] uses the dynamic strings data structure of Gawrychowski et al. [10]. The authors also introduced a simpler alternative, called *Data Dependent Trees*, with logarithmic bounds per operation. On the other hand, Axiotis et al. [3] obtained a deterministic $\mathcal{O}(m \text{ polylog } m)$ algorithm by employing a deterministic data structure of Mehlhorn et al. [18] instead. More precisely, their algorithm is output-sensitive and works in $\mathcal{O}(|X^*| \text{ polylog } |X^*|)$ time, where X^* is the set of all attainable subset sums. The authors provided also a very simple, randomized $\mathcal{O}(m \log^2 m)$ algorithm that uses only an elementary prefix sum structure.

A very recent result of Bringmann and Nakos [8] provides near-linear algorithms for computing the sumset $A_1 + \dots + A_n$, for $A_1, \dots, A_n \subseteq \mathbb{Z}_m$. This problem generalizes the Modular Subset Sum: the set of all attainable subset sums can be expressed as a sumset $\{0, x_1\} + \dots + \{0, x_n\}$.

1.1 Our contributions

In this work, we develop a simple tree-based data structure, designed specifically to handle the text problem that arises in the algorithms for Modular Subset Sum. Our data structure, which we call a *shift-tree*, maintains a string s under the following operations:

- (i) change a single character of s ;
- (ii) cyclically shift s by k positions;
- (iii) given another string t with its corresponding shift-tree, and an interval $[a; b]$, list all positions in $[a; b]$ where strings s and t differ.

We provide two variants of the data structure: a hashing-based one, and a deterministic one with slightly worse time complexity (by $\alpha(n)$, where α is the inverse Ackermann function). By applying shift-trees to the Modular Subset Sum problem, we obtain the following algorithms:

► **Theorem 1.** *There exists an algorithm that returns all attainable modular subset sums of a multiset of integers from \mathbb{Z}_m with high probability, in time $\mathcal{O}(m \log m)$ and space $\mathcal{O}(m)$.*

► **Theorem 2.** *There exists a deterministic algorithm that returns all attainable modular subset sums of a multiset of integers from \mathbb{Z}_m in time $\mathcal{O}(m \log m \cdot \alpha(m))$ and space $\mathcal{O}(m)$.*

The first variant is Monte-Carlo randomized and matches the runtime of Las-Vegas algorithm by Cardinal and Iacono [9]. The second variant is fully deterministic and improves upon the result of Axiotis et al. [3]. Our algorithms are offline as they process the input elements in specific order to achieve their running times.

Although we provide a detailed analysis only for Monte-Carlo randomized and deterministic shift-trees, it is also possible to obtain a Las-Vegas implementation of the data structure. Such an implementation automatically leads to a Las-Vegas algorithm for Modular Subset Sum that truly matches the runtime obtained in [9]. We outline this approach in Remark 8.

Sketch of the shift-tree data structure

We now explain the high-level idea behind our data structure. The shift-tree is a perfect binary tree built upon some string s . The leaves of the tree store the consecutive letters of string s . Since the tree is perfect, the length of string s is required to be a power of two. The inner nodes correspond to substrings of s formed from underlying leaves and store their hashes. The hashes can be updated in logarithmic time after changing a single character of s .

Consider two shift-trees T_1 and T_2 built for strings s_1 and s_2 respectively, such that $|s_1| = |s_2| = m$. We can find all positions where s_1 and s_2 differ by descending from the roots of both trees simultaneously. We compare hashes in the roots and proceed recursively with the children if the hashes differ. Assuming there is no hash collision, we end up in leaves corresponding to positions where s_1 and s_2 differ. Such a procedure will take $\mathcal{O}(k \log m)$ time, where k is the number of differences.

The tricky operation is the cyclic shift of the maintained string. A naive approach would be to simply rebuild the whole tree in linear time. We improve this by noticing that some parts of the tree can be reused. Assume that the string has length m and we want to shift the string by 2^j . Such operation is equivalent to shifting subtrees of size 2^j by 1, what can be done by changing links to children on the appropriate level of the shift-tree. After such modification, hashes on higher levels still need to be updated, but not the hashes in the moved subtrees. This yields a total time of $\mathcal{O}(m/2^j)$ for a cyclic shift by 2^j , and it can be easily extended to shifts of form $k2^j$. Even though it seems like a subtle improvement, it is enough to obtain a fast algorithm for Modular Subset Sum.

To make the shift-trees deterministic, we replace hashes with *tags*. Tags are identifiers associated with strings, but unlike hashes, they are not unique: one string can be represented by multiple tags. Each time a node is updated it receives a new tag. We propagate the information about tags that represent the same strings lazily while searching for differences. More specifically, if the tags are not known to be equal, the search procedure always recurs. If the recursion was unnecessary, we know about it upon return and we can memorize that the respective tags were equivalent.

Sketch of the algorithm for Modular Subset Sum

Our algorithm follows the ideas of [3, 9]. We simulate Bellman's algorithm faster. We iteratively compute the sets of new attainable subset sums C_i after adding the i -th element. More precisely, $C_i = S_i \setminus S_{i-1} = (S_{i-1} + x_i) \setminus S_{i-1}$. The key idea is to notice that instead of computing C_i , we can compute the symmetric difference $D_i = (S_{i-1} + x_i) \triangle S_{i-1}$, and then reduce it to C_i , because $|D_i| = 2|C_i|$.

Let $s_i \in \{0, 1\}^m$ be the characteristic vector of the set S_i , i.e. $s_i[j] = 1$ iff $j \in S_i$. The problem of finding the set D_{i+1} is then reduced to the problem of finding differences between the string s_i and its cyclic shift. We apply shift-trees to solve this problem efficiently. The shift-tree requires the length of the string to be a power of two, so we assume that $m = 2^k$ for now (we show how to get rid of this assumption in section 5). Consider two shift-trees T_1 and T_2 built for string s_i and its cyclic shift respectively. We simulate the Bellman's algorithm step as follows:

- (i) adjust the cyclic shift of T_2 ;
- (ii) find the set D_i by comparing T_1 and T_2 ;
- (iii) update the trees with new attainable subset sums.

The bottleneck of the algorithm are the adjustments of cyclic shift of T_2 : if elements are processed in arbitrary order, the total complexity of shift operations can be $\mathcal{O}(m^2)$. In section 4, we show that if elements are processed in a bit-reversal order, then the shift operations amortize to $\mathcal{O}(m \log m)$.

1.2 Preliminaries

We introduce the following notation for strings. We use the same notation for other sequences.

► **Definition 3.** Given a string $s = c_0 \dots c_{n-1}$, we refer to c_i as $s[i]$ and to substring $c_i \dots c_j$ as $s[i : j]$.

► **Definition 4.** Given a string s and $k \in \mathbb{Z}$, we denote by s^{+k} and s^{-k} the cyclic shift of s by k positions to the right and left respectively. In other words, for every $i \in \{0, \dots, |s| - 1\}$:

$$s[i] = s^{+k}[(i + k) \bmod |s|] = s^{-k}[(i - k) \bmod |s|]$$

2 Shift-trees

2.1 Overview

We introduce *shift-trees*, a variant of the segment tree data structure. A *shift-tree* T maintains a string s of length $m = 2^n$ over an alphabet Σ and supports the following operations:

- $T.\text{INIT}(s)$: Initialize the data structure with string s .
- $T.\text{SET}(i, x)$: Given an index $i \in \{0, \dots, |s| - 1\}$ and a letter $x \in \Sigma$, change $s[i]$ to x .
- $T.\text{SHIFT}(k)$: Given an offset $k \in \mathbb{Z}$, replace s with s^{+k} , i.e. cyclically shift the string s by k positions to the right.
- $T.\text{DIFF}(Q, a, b)$: Given another shift-tree Q representing a string q such that $|s| = |q|$, list all differences between $s[a : b]$ and $q[a : b]$, i.e. return the list L of all integers x such that $a \leq x \leq b$ and $s[x] \neq q[x]$.

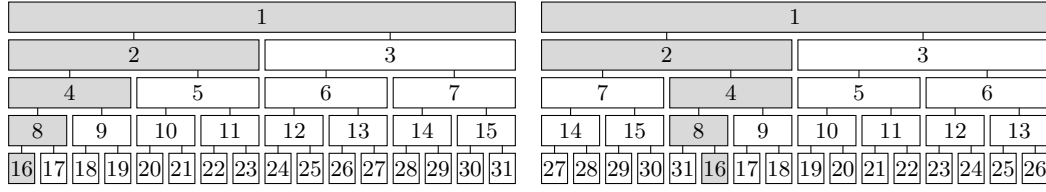
In this section, we describe a hashing-based version of the data structure, which uses $\mathcal{O}(m)$ memory and supports these operations in the following time complexities:

- INIT : $\mathcal{O}(m)$;
- SET : $\mathcal{O}(\log m)$;
- $\text{SHIFT}(k)$: $\mathcal{O}(m/2^j)$, where j is the largest integer such that $2^j \mid k$;
- DIFF : $\mathcal{O}((d + 1) \log m)$, where $d = |L|$ is the number of differences.²

In section 3, we present a variant that is fully deterministic, but achieves a slightly worse time complexity (by $\alpha(m)$, where α is the inverse Ackermann function).

We require an integer alphabet Σ of size $\mathcal{O}(\text{poly}(m))$. We use a standard Rabin-Karp rolling hash function [15]: we choose a sufficiently big prime p and an integer $r \in \mathbb{Z}_p$, where r is chosen uniformly at random. The hash of a string s is defined as $h(s) = \sum_{i=0}^{|s|-1} s[i] \cdot r^i \bmod p$. We assume that $\Sigma \subseteq \mathbb{Z}_p$, so $h(x) = x$ for $x \in \Sigma$. If hashes of two strings of the same length are equal, then the strings are equal with high probability. Moreover, given hashes of some strings s_1 and s_2 , one can compute hash of their concatenation using the following identity: $h(s_1 s_2) = h_1 + h_2 \cdot r^{|s_1|}$. To enable constant-time computation of this formula, we precompute powers of r up to r^m . We use these properties extensively in our data structure.

² By writing $d + 1$ in the complexity, we mean that the runtime of DIFF operation is $\mathcal{O}(\log m)$ if $d = 0$.

(a) $\Delta = 0000_2 = 0$.(b) $\Delta = 0101_2 = 5$.■ **Figure 1** Shift-tree node numbering for different values of Δ .

2.2 Structure

Let s be the string maintained by the data structure and let $|s| = 2^n$. The shift-tree is a perfect binary tree built upon the string s . The leaves of the tree store the consecutive letters of string s , with the leftmost one corresponding to $s[0]$ and the rightmost one corresponding to $s[|s| - 1]$. The inner nodes correspond to substrings of s formed from underlying leaves and store hashes to enable their fast comparison. By $\text{level}(v)$ we denote the distance from the node v to the root node. The root node has level 0 and the leaves have level n . There are 2^k nodes on the k -th level.

We now provide a compact $\mathcal{O}(m)$ memory representation of the data structure that enables us to achieve the desired complexity of SHIFT operation. The only data stored in memory is an array of hashes $H[1 : 2^{n+1} - 1]$ and a single integer $\Delta \in \{0, \dots, 2^n - 1\}$. The values in $H[2^n : 2^{n+1} - 1]$ correspond to the leaves and are letters of the represented string. The value of Δ defines a cyclic shift of leaf indices, i.e. the j -th leftmost leaf of the tree has index $(j - \Delta) \bmod 2^n + 2^n$.

The tree structure is defined implicitly based on the value of Δ as follows. The nodes of the tree are numbered from 1 to $2^{n+1} - 1$. The nodes on the k -th level are numbered from 2^k to $2^{k+1} - 1$. In particular, the root node has index 1 and the leaves have indices from 2^n to $2^{n+1} - 1$. We first introduce the following auxiliary function:

$$\text{skew}(k) = \left\lfloor \frac{\Delta}{2^{n-k}} \right\rfloor \bmod 2$$

Note that floor division by a power of two is equivalent to right bitwise shift, so value of $\text{skew}(k)$ is simply $(n - k)$ -th least significant bit of Δ . Let i be an inner node and let $k = \text{level}(i)$. We define the children of node i as follows:

$$\begin{aligned} \text{left}(i) &= (2i - \text{skew}(k + 1)) \bmod 2^{k+1} + 2^{k+1} \\ \text{right}(i) &= (2i + 1 - \text{skew}(k + 1)) \bmod 2^{k+1} + 2^{k+1} \end{aligned}$$

We also define the parent of node $i \neq 1$ at level k .

$$\text{parent}(i) = \left\lfloor \frac{(i + \text{skew}(k)) \bmod 2^k + 2^k}{2} \right\rfloor$$

The *left*, *right* and *parent* functions can be implemented in constant time. The following lemma and corollary summarize the properties of a tree structure defined as above.

► **Lemma 5.** *The functions left, right and parent define a perfect binary tree, such that the indices of nodes on the k -th level, when ordered from left to right, form a sequence $(2^k, \dots, 2^{k+1} - 1)^{+\lfloor \Delta/2^{n-k} \rfloor}$.*

Proof. We prove the lemma by induction on the level. The condition is satisfied for the 0-th level, which contains only the root node with index 1. Assume now that the condition is satisfied for the k -th level, i.e. the nodes on the k -th level form a sequence $(2^k, \dots, 2^{k+1} - 1)^{+\delta}$, where $\delta = \lfloor \Delta/2^{n-k} \rfloor$. By substituting indices of children in this sequence, we obtain the following sequence for the $(k+1)$ -st level:

$$(\text{left}(2^k), \text{right}(2^k), \dots, \text{left}(2^{k+1} - 1), \text{right}(2^{k+1} - 1))^{+2\delta}$$

The shift is now 2δ , because each element has been replaced by two elements. We want to prove that this is exactly the sequence $(2^{k+1}, \dots, 2^{k+2} - 1)^{+\lfloor \Delta/2^{n-k-1} \rfloor}$. The shift $\lfloor \Delta/2^{n-k-1} \rfloor$ can be rewritten as $2\delta + x$, where $x = \text{skew}(k+1) = \lfloor \Delta/2^{n-k-1} \rfloor \bmod 2$. We now simplify the equation:

$$\begin{aligned} (\text{left}(2^k), \text{right}(2^k), \dots, \text{left}(2^{k+1} - 1), \text{right}(2^{k+1} - 1))^{+2\delta} &\stackrel{?}{=} (2^{k+1}, \dots, 2^{k+2} - 1)^{+2\delta+x} \\ (\text{left}(2^k), \text{right}(2^k), \dots, \text{left}(2^{k+1} - 1), \text{right}(2^{k+1} - 1)) &\stackrel{?}{=} (2^{k+1}, \dots, 2^{k+2} - 1)^{+x} \\ (\text{left}(2^k) - 2^{k+1}, \text{right}(2^k) - 2^{k+1}, \dots, \text{right}(2^{k+1} - 1) - 2^{k+1}) &\stackrel{?}{=} (0, \dots, 2^{k+1} - 1)^{+x} \end{aligned}$$

After substituting the values of *left* and *right*, and simplifying, we obtain the following:

$$((0 - x) \bmod 2^{k+1}, (1 - x) \bmod 2^{k+1}, \dots, (2^{k+1} - 1 - x) \bmod 2^{k+1}) \stackrel{?}{=} (0, \dots, 2^{k+1} - 1)^{+x}$$

The obtained equation trivially satisfies the definition of cyclic shift by x , and all transformations were equivalent. This completes the induction.

We complete the proof by showing that *parent* function is well-defined. Consider an inner node on the k -th level with index $2^k + i$. It is enough to show that it is parent of its children. After substituting and simplifying the formulas, we get the desired result:

$$\begin{aligned} \text{parent}(\text{left}(2^k + i)) &= \left\lfloor \frac{2i + 2^{k+1}}{2} \right\rfloor = 2^k + i \\ \text{parent}(\text{right}(2^k + i)) &= \left\lfloor \frac{2i + 1 + 2^{k+1}}{2} \right\rfloor = 2^k + i \end{aligned} \quad \blacktriangleleft$$

► **Corollary 6.** *The functions *left*, *right* and *parent* define a perfect binary tree such that:*

- (a) *nodes on the k -th level have indices from 2^k to $2^{k+1} - 1$, for each valid k ;*
- (b) *the indices of leaves, when ordered from left to right, form a sequence $(2^n, \dots, 2^{n+1} - 1)^{+\Delta}$;*
- (c) *the structure of subtrees rooted at the k -th level depends only on $\Delta \bmod 2^{n-k}$.*

Proof. The first two properties follow instantly from the lemma 5. For the property (c), notice that the links on these levels depend only on the values $\text{skew}(k+1), \dots, \text{skew}(n)$. These values are exactly the $n - k$ least significant bits of Δ . ◀

2.3 Invariant

Let s be the string maintained by the data structure and let $|s| = 2^n$. We define the string associated with a node i recursively as follows:

$$\text{str}(i) = \begin{cases} s^{-\Delta}[i - 2^n] & \text{for } i \geq 2^n \text{ (i.e. } i \text{ is leaf node)} \\ \text{str}(\text{left}(i)) \text{str}(\text{right}(i)) & \text{for } i < 2^n \text{ (i.e. } i \text{ is inner node)} \end{cases}$$

By corollary 6b, the k -th letter of string s is associated with the k -th leftmost leaf. It follows that $s = \text{str}(1)$, i.e. string associated with the root node is s . We maintain the following invariant:

► **Invariant 7.** For each node i the following holds: $H[i] = h(\text{str}(i))$.

The invariant ensures that each node stores a hash of its associated string. We use this property to implement the DIFF operation in required time complexity.

2.4 Operations

Let s be the string maintained by the data structure and let $|s| = m = 2^n$. We first define an $\text{UPDATE}(i)$ primitive that is used by all operations that modify the data structure. The UPDATE procedure simply recalculates the hash of an inner node i based on hashes of its children. This can be done in constant time using basic modular arithmetic, if appropriate powers of r are precomputed.

■ **Algorithm 1** The UPDATE procedure.

```

1: function  $\text{UPDATE}(i)$ 
2:    $H[i] \leftarrow (H[\text{left}(i)] + H[\text{right}(i)] \cdot r^{|\text{str}(\text{left}(i))|}) \bmod p$ 

```

Init. We initialize Δ with 0 and leaves with the letters of the input string s . Specifically, we set $H[2^n + i] = s[i]$ for each $i \in \{0, \dots, 2^n - 1\}$, because letter $s[i]$ is associated with the node $2^n + i$. We then compute all the hashes by calling an UPDATE on the remaining nodes, beginning at the bottom of the tree. Overall, the INIT operation updates $\mathcal{O}(m)$ nodes and runs in $\mathcal{O}(m)$ time.

Set. Assume that we change $s[i]$ to x . Let $j = (i - \Delta) \bmod 2^n + 2^n$. Notice that, $\text{str}(j) = s^{-\Delta}[j - 2^n] = s[(j + \Delta) \bmod 2^n] = s[i]$. In order to fix the invariant, we set $H[j] = x$ and update hashes of all the ancestors of j . The tree has $\mathcal{O}(\log m)$ levels, so the total runtime of SET operation is $\mathcal{O}(\log m)$.

■ **Algorithm 2** INIT operation.

```

1: function  $\text{INIT}(s)$ 
2:    $\Delta \leftarrow 0$ 
3:   for  $i \leftarrow 0, \dots, 2^n - 1$  do
4:      $H[2^n + i] \leftarrow s[i]$ 
5:   for  $i \leftarrow 2^n - 1, \dots, 1$  do
6:      $\text{UPDATE}(i)$ 

```

■ **Algorithm 3** SET operation.

```

1: function  $\text{SET}(i, x)$ 
2:    $j \leftarrow (i - \Delta) \bmod 2^n + 2^n$ 
3:    $H[j] \leftarrow x$ 
4:   while  $j \neq 1$  do
5:      $j \leftarrow \text{parent}(j)$ 
6:      $\text{UPDATE}(j)$ 

```

Shift. Assume that we apply a right cyclic shift by k positions to s . Let j be the largest integer such that $2^j \mid k$. In order to fix the invariant, we first set Δ to $(\Delta + k) \bmod 2^n$. Notice that the invariant is now satisfied for leaves. Moreover, by corollary 6c the structure of subtrees rooted at level $n - j$ didn't change, so the invariant is also satisfied for levels $n - j, \dots, n$. It remains to update hashes on the remaining $n - j$ levels by calling the UPDATE procedure on their nodes. Overall, $\mathcal{O}(2^{n-j})$ nodes are updated and the SHIFT operation runs in $\mathcal{O}(m/2^j)$ time.³

³ Provided algorithm requires computation of $j = \max\{d \in \mathbb{N} : 2^d \mid k\}$. In practice, it is sufficient to compute 2^j instead of computing j directly. This can be done in constant time using the following bit-hack: $k \& \sim(k-1)$.

■ **Algorithm 4** SHIFT operation.

```

1: function SHIFT( $k$ )
2:   if  $k \bmod 2^n = 0$  then
3:     return
4:    $j \leftarrow \max\{d \in \mathbb{N} : 2^d \mid k\}$ 
5:    $\Delta \leftarrow (\Delta + k) \bmod 2^n$ 
6:   for  $i \leftarrow 2^{n-j} - 1, \dots, 1$  do
7:     UPDATE( $i$ )

```

Diff. Assume we look for differences between the strings maintained by the trees T and Q in interval $[a; b]$. We provide a recursive procedure $\text{FINDDIFFERENCES}(T, Q, a, b, i, j, x, y)$ that returns required set of differences between substrings associated with node i of the tree T and node j of the tree Q . The procedure additionally tracks an interval $[x; y]$ that is associated with both nodes, i.e. $T.\text{str}(i) = T.s[x : y]$ and $Q.\text{str}(j) = Q.s[x : y]$.

The FINDDIFFERENCES procedure works as follows. If $[a; b] \cap [x; y] = \emptyset$ then procedure returns empty set instantly, because we only look for differences in the interval $[a; b]$. If hashes of nodes i and j are equal then the substrings are equal w.h.p., so the procedure returns no differences as well. Otherwise, there is at least one difference between the strings associated with nodes i and j . If the nodes are leaves, then we report the difference. If the nodes are inner nodes, the procedure is invoked recursively on left and right children.

The DIFF operation simply calls FINDDIFFERENCES on roots of the trees T and Q . The procedure will return all the required differences as long as there is no hash collision.

■ **Algorithm 5** The DIFF operation.

```

1: function T.DIFF( $Q, a, b$ )
2:   return FINDDIFFERENCES( $T, Q, a, b, 1, 1, 0, 2^n - 1$ )
3:
4: function FINDDIFFERENCES( $T, Q, a, b, i, j, x, y$ )
5:   if  $[a; b] \cap [x; y] = \emptyset$  or  $T.H[i] = Q.H[j]$  then
6:     return  $\emptyset$ 
7:   if  $x = y$  then ▷ The nodes  $i$  and  $j$  are leaves if they represent an unit interval
8:     return  $\{x\}$ 
9:    $z \leftarrow \frac{x+y+1}{2}$  ▷ The left nodes represent  $[x : z - 1]$  and the right nodes represent  $[z : y]$ 
10:   $A \leftarrow \text{FINDDIFFERENCES}(T, Q, a, b, T.\text{left}(i), Q.\text{left}(j), x, z - 1)$ 
11:   $B \leftarrow \text{FINDDIFFERENCES}(T, Q, a, b, T.\text{right}(i), Q.\text{right}(j), z, y)$ 
12:  return  $A \cup B$ 

```

We now argue the complexity of DIFF operation. Let d be the number of differences that were found. Let s_k be the number of FINDDIFFERENCES calls for which the processed nodes were on the k -th level and the procedure recurred. If the procedure recurred, there existed at least one difference between the strings associated with the nodes. We can divide such calls into two categories:

- (a) there is a difference in $[x; y] \cap [a; b]$ that should be reported;
- (b) there is a difference in $[x; y] \setminus [a; b]$ that should be ignored and $[x; y] \cap [a; b] \neq \emptyset$.

The number of calls that belong to the category (a) is bounded by d , i.e. number of reported differences. There are at most 2 calls that belong to the category (b), because the interval $[x; y]$ must contain a or b . It follows that $s_k \leq d + 2$ and $s_0 + \dots + s_{n-1} \leq (d + 2) \cdot n$. We can charge the calls that didn't recur to their parents, so the total running time of DIFF operation is $\mathcal{O}((d + 1) \log m)$.

In order to complete the analysis, we bound the probability that DIFF operation fails to report all differences. Such situation may occur only if nodes processed by FINDDIFFERENCES have different associated strings, but equal hashes. Let $s_1 \neq s_2$ be strings of length k .

$$\Pr[h(s_1) = h(s_2)] = \Pr\left[\sum_{i=0}^{k-1} (s_1[i] - s_2[i])r^i \equiv 0 \pmod{p}\right] \leq \frac{k}{p}$$

The inequality holds, because the sum on the left side is a non-zero polynomial of degree at most k , evaluated in randomly chosen point r . By application of union bound, we obtain that the probability of failure is at most $m \log m / p$. Assuming operations on hashes of size $\mathcal{O}(\log m)$ are taking constant time, we can choose $p = \Theta(\text{poly}(m))$ and obtain high probability of success.

► **Remark 8.** It is also possible to achieve a Las-Vegas implementation of shift-trees. The key observation is that the hash function doesn't need to be associative, i.e. one can hash "subtrees" instead of substrings. This allows us to replace the hash function with any injective mapping $h(x, y)$ from pairs of hashes into new hashes. Only the UPDATE procedure needs to be adapted to compute the hash $H[i]$ of i -th node as $h(H[\text{left}(i)], H[\text{right}(i)])$.

The missing piece is how to implement the mapping $h(x, y)$. This can be done by simply generating it on demand, and storing the mapping in a hashtable. Some garbage collection mechanism (such as reference counting) is required to maintain linear memory usage. This yields a Las-Vegas implementation of shift-trees, with the same expected runtime bounds.

By replacing hashtable with a BST, one can obtain a deterministic implementation of the data structure. Such modification introduces logarithmic runtime overhead. In the next section, we improve upon this by allowing amortization.

3 Deterministic shift-trees

3.1 Overview

We now provide a deterministic variant of the data structure introduced in previous section. Let $m = 2^n$. Assume that we maintain several shift-trees T_1, \dots, T_r associated with strings $s_1, \dots, s_r \in \Sigma^m$ of the same length, allowing comparisons between those strings. Let $K = |s_1| + \dots + |s_r| = mr$, and let T_i be one of the trees. Then, we can do the shift-tree operations on T_i with the following amortized time complexities:

- INIT: $\mathcal{O}(m \cdot \alpha(K))$;
- SET: $\mathcal{O}(\log m \cdot \alpha(K))$;
- SHIFT(k): $\mathcal{O}(m/2^j \cdot \alpha(K))$, where j is the largest integer such that $2^j \mid k$;
- DIFF: $\mathcal{O}((d+1) \log m \cdot \alpha(K))$, where d is the number of differences.

Moreover, the data structures use $\mathcal{O}(K)$ memory overall. The only constraint that we put on alphabet Σ is the support for equality tests. This contrasts with the randomized variant, where an integer alphabet of polynomial size is required.

3.2 Tags

We replace hashing with the concept of *tags*. Tags are simply identifiers associated with strings. Unlike hashes, they are not unique: one string can be represented by multiple tags. Each inner node of the shift-tree stores a tag instead of a hash. A new tag is created every time a node is updated.

We denote the string associated with a tag t by $\text{str}(t)$. The strings associated with tags are not stored in memory. Instead, we maintain an equivalence relation \mathcal{R} over the set of tags used in all the shift-trees. The relation \mathcal{R} satisfies the following property:

► **Invariant 9.** *For each tag a and tag b such that $a \equiv_{\mathcal{R}} b$, the strings $\text{str}(a)$ and $\text{str}(b)$ are equal.*

Note that the inverse doesn't need to hold. In other words, the relation \mathcal{R} partially captures the equality relation between strings associated with tags. The relation is refined during operations on the shift-trees using the following operations:

- **NEWTAG()**: Create a new tag x with its own singleton equivalence class, and return x .
- **FIND(x)**: Given a tag x , return identifier of its equivalence class.
- **UNION(x, y)**: Given tags x and y , union their equivalence classes (i.e. insert $x \equiv_{\mathcal{R}} y$ and close the relation transitively).
- **DELETETAG(x)**: Given a tag x , remove it from its equivalence class and free its memory.

In order to support these operations efficiently, we represent the equivalence classes of \mathcal{R} using a union-find data structure. A simple extension of standard disjoint-set forest implementation with support for element removal has been proposed by Kaplan et al. in [13].

► **Theorem 10** ([13]). *There exists a data structure that maintains an equivalence relation \mathcal{R} using linear memory under operations **FIND**, **UNION** and **DELETETAG** in amortized $\mathcal{O}(\alpha(n))$ time, and **NEWTAG** in $\mathcal{O}(1)$ time, where n is the number of maintained elements.*

Their approach is based on lazy deletions: elements to be deleted are marked and the union-find trees are rebuilt if the fraction of marked elements is greater than half. More involved approaches with constant time deletions have been known in literature [2, 6], but such improvement doesn't change the amortized time complexity of our data structure.

The idea to use union-find data structure for detecting mismatches has been already proposed by Gawrychowski et al. in [11].

► **Remark.** In general, the **DELETETAG** operation is not only useful for space optimization. If unused tags are not removed, the complexity of operations is dependent on the total number of **NEWTAG** calls, which can be large. This is not an issue if only $\mathcal{O}(\text{poly}(K))$ tags are created in total, because $\mathcal{O}(\alpha(\text{poly}(K))) = \mathcal{O}(\alpha(K))$.

3.3 Operations

We now adapt the **UPDATE** procedure and **DIFF** operation to work with tags. The **INIT**, **SET** and **SHIFT** operations use the **UPDATE** primitive and don't require changes.

Update. We simply create a new tag for the updated node. If the node already contains a tag (i.e. the **UPDATE** is called after initialization), we delete it in order to maintain linear memory usage. The newly created tag is in a singleton equivalence class of \mathcal{R} , so it trivially satisfies the invariant.

■ **Algorithm 6** The **UPDATE** procedure.

```

1: function UPDATE( $i$ )
2:   if  $H[i] \neq \text{null}$  then
3:     DELETETAG( $H[i]$ )
4:    $H[i] \leftarrow \text{NEWTAG}()$ 

```

Diff. We adapt the `FINDDIFFERENCES` procedure as follows. Assume that we are looking for differences in interval $[a; b]$. Let t_1 and t_2 be tags in compared tree nodes, and $[x; y]$ the interval associated with these nodes. If $t_1 \equiv_{\mathcal{R}} t_2$ then the compared substrings are equal, so we exit instantly. Otherwise, we search for differences recursively in the left and right subtrees. If no differences are found and $[x; y] \subseteq [a; b]$, we know that $\text{str}(t_1) = \text{str}(t_2)$, so we can safely add $t_1 \equiv_{\mathcal{R}} t_2$ to relation \mathcal{R} via `UNION`(t_1, t_2).

■ **Algorithm 7** The `FINDDIFFERENCES` procedure.

```

1: function FINDDIFFERENCES( $T, Q, a, b, i, j, x, y$ )
2:   if  $[a; b] \cap [x; y] = \emptyset$  then                                ▷ Check if we are outside of the search interval
3:     return  $\emptyset$ 
4:   if  $x = y$  then                                                ▷ The nodes  $i$  and  $j$  are leaves if they represent an unit interval
5:     if  $T.H[i] \neq Q.H[j]$  then                                    ▷ The leaves contain letters – compare them directly
6:       return  $\{x\}$ 
7:     else
8:       return  $\emptyset$ 
9:   if  $\text{FIND}(T.H[i]) = \text{FIND}(Q.H[j])$  then                        ▷ The inner nodes contain tags – use  $\mathcal{R}$ .
10:    return  $\emptyset$                                                   ▷  $T.H[i] \equiv_{\mathcal{R}} Q.H[j]$  holds, so the strings are equal
11:   $z \leftarrow \frac{x+y+1}{2}$  ▷ The left nodes represent  $[x; z-1]$  and the right nodes represent  $[z; y]$ 
12:   $A \leftarrow \text{FINDDIFFERENCES}(T, Q, a, b, T.\text{left}(i), Q.\text{left}(j), x, z-1)$ 
13:   $B \leftarrow \text{FINDDIFFERENCES}(T, Q, a, b, T.\text{right}(i), Q.\text{right}(j), z, y)$ 
14:  if  $A \cup B = \emptyset$  and  $[x; y] \subseteq [a; b]$  then
15:    UNION( $T.H[i], Q.H[j]$ )
16:  return  $A \cup B$ 

```

We now briefly address the correctness of the `FINDDIFFERENCES` procedure. Observe that if the condition $[a; b] \cap [x; y] \neq \emptyset$ holds then:

- (i) the invariant 9 guarantees that the procedure will recur if substrings are not equal;
- (ii) the procedure returns a difference for leaves iff their corresponding characters differ.

It follows by an easy induction on the level that the procedure returns all positions in $[a; b]$ where the strings differ and nothing more. Moreover, the procedure doesn't break the invariant when modifying the relation \mathcal{R} : if the condition in line 14 is true, then there are no differences between compared substrings.

3.4 Running time

Let K be the sum of lengths of strings maintained by all shift-trees and let $m = 2^n$ be the length of each string. We first note that the number of elements maintained by relation \mathcal{R} never exceeds the total number of nodes in shift-trees, which is $\mathcal{O}(K)$, so any operation on \mathcal{R} works in amortized $\mathcal{O}(\alpha(K))$ time.

We now argue the amortized running time of the `UPDATE` and `DIFF` operations. Let the actual cost of the i -th operation be a number c_i of operations on the relation \mathcal{R} . Let q_i be the number of equivalence classes of relation \mathcal{R} after i operations. We define potential Φ_i to be $9q_i$. Clearly, $\Phi_i \geq \Phi_0 = 0$. The amortized cost of the i -th operation is $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$.

The actual cost of an `UPDATE` operation is $c_i \leq 2$. The operation creates at most one new equivalence class, thus the amortized cost is $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} \leq 2 + 9 \cdot 1 = 11$. It follows that the amortized time complexity of an `UPDATE` operation is $\mathcal{O}(\alpha(K))$.

To estimate the amortized cost of `DIFF` operation, we consider the `FINDDIFFERENCES` calls that recurred. We say that the call is *wasted* if the compared strings are equal, but comparison of tags reported that they are not. Otherwise, if the compared strings are not

equal and the procedure recurred, we say that the call is *required*. Let w be the number of wasted calls and r be the number of required calls. We can charge the calls that didn't recur to their parents, so the total number of `FINDDIFFERENCES` calls is bounded by $3(r + w)$. Each call does at most three operations on relation \mathcal{R} , so the cost of `DIFF` operation is $c_i \leq 9(r + w)$.

Let d be the number of differences that have been found. We can bound the number of required calls r by $\mathcal{O}((d + 1) \log m)$, the same way as in hashing-based shift-trees. We now focus our attention on the wasted calls. Assume that we are looking for differences in interval $[a; b]$. Let w' be the number of wasted calls such that the interval associated with compared nodes is contained within $[a; b]$. Notice that upon return, each such call unions two different equivalence classes, thus decreasing the potential by 9. It follows that `DIFF` operation reduces the potential in total by $9w'$. On the other hand, the intervals associated with the remaining wasted calls must contain a or b , so there are at most 2 such calls for each tree level. It follows that the number of all wasted calls is bounded by $w' + 2 \log m$. The amortized cost of `DIFF` operation is then:

$$\hat{c}_i \leq 9r + 9w - 9w' \leq 9r + 18 \log m = \mathcal{O}((d + 1) \log m)$$

and the amortized time complexity is $\mathcal{O}((d + 1) \log m \cdot \alpha(K))$.

We complete analysis by providing amortized running time of `INIT`, `SET` and `SHIFT` operations. The `INIT` operation calls `UPDATE` operation $\mathcal{O}(m)$ times, so its amortized running time is $\mathcal{O}(m \cdot \alpha(K))$. By the same argument we obtain the required amortized complexities for `SET` and `SHIFT` operation.

4 Traversing all cyclic shifts

In this section, we consider a problem of going over all the cyclic shifts of the shift-tree efficiently, in some order. This means that we want to consider all shifts $s^{+\sigma(0)}, \dots, s^{+\sigma(|s|-1)}$, for σ being some permutation of $\{0, \dots, |s| - 1\}$. This requires invoking `SHIFT`($\sigma(i) - \sigma(i - 1)$) for $i = 1, 2, \dots, |s| - 1$, assuming the shift-tree initially represents $s^{+\sigma(0)}$. We claim that there exists a permutation σ such that the total complexity of these operations amortizes to $\mathcal{O}(m \log m)$ time. For simplicity, we consider the hashing-based shift-trees; the complexity for deterministic variant is just multiplied by $\alpha(K)$. This technique is crucial for our Modular Subset Sum algorithm and might be used for other problems, where the order of operations doesn't matter.

The time complexity of a single `SHIFT` operation depends heavily on the value of shift. Recall that the running time of `SHIFT`(k) is $\mathcal{O}(m/2^j)$, where j is the largest integer such that $2^j \mid k$, and $m = 2^n$ is the size of the shift-tree. For example, a shift by 1 requires rebuilding the entire tree, while shift by $m/2$ takes constant time. It means that the complexity of going over all the cyclic shifts heavily depends on the permutation σ .

It turns out that a good permutation is a bit-reversal permutation, which we define as follows. We denote the bit-reverse of n -bit number j by $\text{bitrev}_n(j)$, i.e. if $j = \sum_{i=0}^{n-1} c_i 2^i$ then $\text{bitrev}_n(j) = \sum_{i=0}^{n-1} c_i 2^{n-i-1}$. We say that σ is a *bit-reversal permutation* if $\sigma(i) = \text{bitrev}_n(i)$.

► **Lemma 11.** *Let σ be a bit-reversal permutation of length $m = 2^n$ and T a shift-tree of length $\mathcal{O}(m)$. The sequence of operations $T.\text{SHIFT}(\sigma_i - \sigma_{i-1})$ for $i = 1, \dots, m - 1$ takes total time $\mathcal{O}(m \log m)$.*

Proof. Let $\delta_i = \sigma_i - \sigma_{i-1}$ and consider a single $\text{SHIFT}(\delta_i)$ operation. Let j be the largest integer such that $2^j \mid \delta_i$. The complexity of this operation is then $\mathcal{O}(m/2^j) = \mathcal{O}(2^{n-j})$. If $2^j \mid \delta_i$ and $2^{j+1} \nmid \delta_i$ then j is the least significant bit that is different between σ_i and σ_{i-1} . Since σ_i is a bit-reverse of i , it means that $n-j-1$ is the most significant bit that is different between i and $i+1$. Such situation happens only if $i+1$ is of form $k \cdot 2^{n-j-1}$, where k is odd. There are 2^j such numbers in range $[1; 2^n - 1]$. It follows that shifts for a given value of j take overall $\mathcal{O}(2^{n-j} \cdot 2^j) = \mathcal{O}(2^n)$ time. There are $\mathcal{O}(n)$ possible values of j , so the whole sequence of shifts takes $\mathcal{O}(2^n \cdot n) = \mathcal{O}(m \log m)$ time. ◀

5 Modular Subset Sum

In this section, we provide an algorithm for the Modular Subset Sum problem that uses the shift-tree data structure. Let $X = \{x_1, \dots, x_n\}$ be a multiset of integers from \mathbb{Z}_m . Our algorithm computes the set $X^* \subseteq \mathbb{Z}_m$ such that $k \in X^*$ if and only if there exists a subset of X that sums to the value k modulo m . We assume that the input multiset X is provided in a compact form: as a list of $\mathcal{O}(m)$ distinct elements along with their multiplicities.

The algorithm is based on the so-called Bellman's iteration. Consider sets $S_0, \dots, S_n \subseteq \mathbb{Z}_m$ such that $S_0 = \{0\}$ and $S_i = S_{i-1} \cup (S_{i-1} + x_i)$, where $S_{i-1} + x_i = \{a + x_i : a \in S_{i-1}\}$. It is easy to see that the set S_i is a set of all attainable subset sums of $\{x_1, \dots, x_i\}$ and the final result is $X^* = S_n$. In order to compute the set S_i from S_{i-1} , it is sufficient to find the set $C_i = S_i \setminus S_{i-1} = (S_{i-1} + x_i) \setminus S_{i-1}$. If one can compute the set C_i in time $\mathcal{O}(|C_i| \cdot f(m))$, then the set X^* can be computed in total time $\mathcal{O}(m \cdot f(m))$.

The key idea of [4] is to notice that instead of computing $C_i = (S_{i-1} + x_i) \setminus S_{i-1}$, we can compute the symmetric difference $D_i = (S_{i-1} + x_i) \triangle S_{i-1}$. Computing the set D_i doesn't break the time complexity, because its size is only two times larger than C_i . We can now interpret problem of finding the set D_i as a text problem [3, 9]. Let $s_i \in \{0, 1\}^m$ be the characteristic vector of the set S_i , i.e. $s_i[j] = 1$ iff $j \in S_i$. The problem of finding the set D_{i+1} is then reduced to problem of finding differences between the strings s_i and $s_i^{+x_{i+1}}$. The set D_{i+1} is exactly the set of indices, where these strings differ.

We now describe our algorithm. Let S be the set of all subset sums attainable using elements processed so far, and let s be its characteristic vector. Initially, the set S contains only 0. We maintain the characteristic vector s and its cyclic shift using two shift-trees, T_1 and T_2 respectively. The length of a string maintained by a shift-tree is required to be a power of two, but that may not be the case with the string s . We address this issue in the following way. Let L be the smallest power of two such that $L \geq 2m$. We consider the following auxiliary strings:

$$\begin{aligned} s' &= s0^{L-m} \\ s'' &= s0^{L-2m}s \end{aligned}$$

The strings s' and s'' have length L , which is a power of two. Moreover, the string s'' has the following property: the string s^{+d} is a prefix of $(s'')^{+d}$, for any $d \in \{0, \dots, m\}$. This property allows us to find differences between s and s^{+d} by comparing a prefix of s' with a prefix of $(s'')^{+d}$.

The tree T_1 maintains the string s' and the tree T_2 maintains a cyclic shift of the string s'' . Specifically, we traverse all cyclic shifts of T_2 in bit-reversal order, as explained in the previous section. Assume that the current shift of T_2 is x , i.e. the string maintained by T_2 is $(s'')^{+x}$. Let μ be the multiplicity of x in the input multiset X . If $\mu = 0$, then $x \notin X$ and the algorithm proceeds to the next shift. Otherwise, we simulate μ Bellman's iterations for the

element x as follows. The x is contained in the multiset X , so $x \in \{0, \dots, m-1\}$. It implies that we can find the set of differences D between s and s^{+x} by comparing a prefix of s' with a prefix of $(s'')^{+x}$. This is done using DIFF operation on T_1 and T_2 . We then update the set of attainable subset sums S and both shift-trees appropriately. If no differences were found, we skip the rest of iterations for element x .

Every element of X corresponds to some cyclic shift, so all elements will be processed if all the cyclic shifts are considered. The set S is then the set of all attainable subset sums for X . We provide the pseudocode as Algorithm 8. In the pseudocode, we denote the multiplicity of element x in the set X by $\mu_X(x)$.

■ **Algorithm 8** The MODULARSUBSETSUM algorithm.

```

1: function MODULARSUBSETSUM( $X, m$ )
2:    $S = \{0\}$ 
3:    $k \leftarrow \min\{d \in \mathbb{N} : 2^d \geq 2m\}$ 
4:    $L \leftarrow 2^k$ 
5:    $s_0 \leftarrow 10^{m-1}$  ▷ The characteristic vector of  $S_0 = \{0\}$ 
6:   Initialize shift-tree  $T_1$  with  $s_0 0^{L-m}$ 
7:   Initialize shift-tree  $T_2$  with  $s_0 0^{L-2m} s_0$ 
8:   for  $i \leftarrow 1, \dots, L-1$  do
9:      $x \leftarrow \text{bitrev}_k(i)$ 
10:     $T_2.\text{SHIFT}(x - \text{bitrev}_k(i-1))$  ▷ Now  $T_2$  represents the string  $(s'')^{+x}$ 
11:    for  $j \leftarrow 1, \dots, \mu_X(x)$  do
12:       $D \leftarrow T_1.\text{DIFF}(T_2, 0, m-1)$  ▷  $D$  is the set of all differences between  $s$  and  $s^{+x}$ 
13:      if  $D = \emptyset$  then ▷ Skip the rest of iterations for  $x$  if no differences were found
14:        break
15:      for  $d \in D \setminus S$  do
16:         $S \leftarrow S \cup \{d\}$ 
17:         $T_1.\text{SET}(d, 1)$ 
18:         $T_2.\text{SET}((d+x) \bmod L, 1)$ 
19:         $T_2.\text{SET}((d+x-m) \bmod L, 1)$ 
20:  return  $S$ 

```

We now analyse the running time of the algorithm. We assume the hashing-based shift-trees are used; the complexity for deterministic variant is just multiplied by $\alpha(m)$. The initialization of shift-trees takes $\mathcal{O}(m)$ time. The value of $\text{bitrev}_k(i)$ can be computed naively bit by bit in $\mathcal{O}(k) = \mathcal{O}(\log m)$ time, which in total takes $\mathcal{O}(m \log m)$ time. Moreover, the total time of all SHIFT operations amortizes to $\mathcal{O}(m \log m)$ time due to lemma 11. We now focus on the total running time of inner loops.

Consider a single Bellman's iteration. The complexity of a DIFF operation is $\mathcal{O}((|D| + 1) \log m)$. The algorithm adds $|D \setminus S| = |D|/2$ new elements to the set S and updates the shift-trees. Each tree update takes $\mathcal{O}(\log m)$ time. In total, a single Bellman's iteration takes $\mathcal{O}((|D| + 1) \log m)$ time.

The sum of sizes of all the sets of differences is at most $2m$. It means that if there are k Bellman's iterations in total, then their total running time is $\mathcal{O}((m+k) \log m)$. The condition in the line 13 ensures that the total number of executed Bellman's iterations is $\mathcal{O}(m)$ by skipping the iterations if the set is empty. It follows that all the iterations take $\mathcal{O}(m \log m)$ time in total.

We arrive at the total time complexity of $\mathcal{O}(m \log m)$. By replacing hashing-based shift-trees with their deterministic variant, we obtain a deterministic algorithm with running time of $\mathcal{O}(m \log m \cdot \alpha(m))$. We recall the theorems that summarize these results:

► **Theorem 1.** *There exists an algorithm that returns all attainable modular subset sums of a multiset of integers from \mathbb{Z}_m with high probability, in time $\mathcal{O}(m \log m)$ and space $\mathcal{O}(m)$.*

► **Theorem 2.** *There exists a deterministic algorithm that returns all attainable modular subset sums of a multiset of integers from \mathbb{Z}_m in time $\mathcal{O}(m \log m \cdot \alpha(m))$ and space $\mathcal{O}(m)$.*

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. doi:10.1137/1.9781611975482.3.
- 2 Stephen Alstrup, Inge Li Gørtz, Theis Rauhe, Mikkel Thorup, and Uri Zwick. Union-find with constant time deletions. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 78–89. Springer, 2005. doi:10.1007/11523468_7.
- 3 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 57–67. SIAM, 2021. doi:10.1137/1.9781611976496.6.
- 4 Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. 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 58–69. SIAM, 2019. doi:10.1137/1.9781611975482.4.
- 5 Richard E Bellman et al. Dynamic programming. *Cambridge Studies in Speech Science and Communication*. Princeton University Press, Princeton, 1957.
- 6 Amir M. Ben-Amram and Simon Yoffe. A simple and efficient union-find-delete algorithm. *Theor. Comput. Sci.*, 412(4-5):487–492, 2011. doi:10.1016/j.tcs.2010.11.005.
- 7 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. 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 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.
- 8 Karl Bringmann and Vasileios Nakos. Fast n-Fold Boolean Convolution via Additive Combinatorics. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.41.
- 9 Jean Cardinal and John Iacono. Modular subset sum, dynamic strings, and zero-sum sets. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 45–56. SIAM, 2021. doi:10.1137/1.9781611976496.5.
- 10 Pawel Gawrychowski, Adam Karczmaz, 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.
- 11 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. 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 5:1–5:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.5.

- 12 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 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASICS*, pages 17:1–17:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.17.
- 13 Haim Kaplan, Nira Shafrir, and Robert Endre Tarjan. Union-find with deletions. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 19–28. ACM/SIAM, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545384>.
- 14 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 15 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.
- 16 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. 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 1062–1072. SIAM, 2017. doi:10.1137/1.9781611974782.68.
- 17 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.
- 18 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.

Hardness of Detecting Abelian and Additive Square Factors in Strings

Jakub Radoszewski ✉ 

University of Warsaw, Poland
Samsung R&D, Warsaw, Poland

Wojciech Rytter ✉ 

University of Warsaw, Poland

Juliusz Straszyński ✉ 

University of Warsaw, Poland

Tomasz Waleń ✉ 

University of Warsaw, Poland

Wiktor Zuba ✉ 

University of Warsaw, Poland

Abstract

We prove 3SUM-hardness (no strongly subquadratic-time algorithm, assuming the 3SUM conjecture) of several problems related to finding Abelian square and additive square factors in a string. In particular, we conclude conditional optimality of the state-of-the-art algorithms for finding such factors.

Overall, we show 3SUM-hardness of (a) detecting an Abelian square factor of an odd half-length, (b) computing centers of all Abelian square factors, (c) detecting an additive square factor in a length- n string of integers of magnitude $n^{\mathcal{O}(1)}$, and (d) a problem of computing a double 3-term arithmetic progression (i.e., finding indices $i \neq j$ such that $(x_i + x_j)/2 = x_{(i+j)/2}$) in a sequence of integers x_1, \dots, x_n of magnitude $n^{\mathcal{O}(1)}$.

Problem (d) is essentially a convolution version of the AVERAGE problem that was proposed in a manuscript of Erickson. We obtain a conditional lower bound for it with the aid of techniques recently developed by Dudek et al. [STOC 2020]. Problem (d) immediately reduces to problem (c) and is a step in reductions to problems (a) and (b). In conditional lower bounds for problems (a) and (b) we apply an encoding of Amir et al. [ICALP 2014] and extend it using several string gadgets that include arbitrarily long Abelian-square-free strings.

Our reductions also imply conditional lower bounds for detecting Abelian squares in strings over a constant-sized alphabet. We also show a subquadratic upper bound in this case, applying a result of Chan and Lewenstein [STOC 2015].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Abelian square, additive square, 3SUM problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.77

Related Version *Full Version:* <https://arxiv.org/abs/2107.09206>

Funding *Jakub Radoszewski:* Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Juliusz Straszyński: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Tomasz Waleń: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Wiktor Zuba: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Acknowledgements The authors warmly thank Paweł Gawrychowski and Tomasz Kociumaka for helpful discussions.



© Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 77; pp. 77:1–77:19

Leibniz International Proceedings in Informatics



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

1 Introduction

Abelian squares. An Abelian square, Ab-square in short (also known as a jumbled square), is a string of the form XY , where Y is a permutation of X ; we say that X and Y are Ab-equivalent. We are interested in factors (i.e., substrings composed of consecutive letters) of a given text string being Ab-squares.

► **Example 1.** The string

0 6 1 0 5 6 5 1 6 1 0 1 1 1 0 6 5 6 5 7 8 6 1 6 5 0 5 1 0 5 6 6 5 0 6 0 3 0 6 5 2

has exactly two Ab-square factors of length 12, shown above (but it has also Ab-squares of other lengths, e.g. 5665, 11, 1111, 011110).

Ab-squares were first studied by Erdős [16], who posed a question on the smallest alphabet size for which there exists an infinite Ab-square-free string, i.e., an infinite string without Ab-square factors. The first example of such a string over a finite alphabet was given by Evdokimov [18]. Later the alphabet size was improved to five by Pleasants [34] and finally an optimal example over a four-letter alphabet was shown by Keränen [27]. Further results on combinatorics of Ab-square-free strings and several examples of their applications in group theory, algorithmic music and cryptography can be found in [26] and references therein. Avoidability of long Ab-squares was also considered [36].

Strings containing Ab-squares were also studied. Motivated by another problem posed by Erdős [16], Entringer et al. [15] showed that every infinite binary string has arbitrarily long Ab-square factors. Fici et al. [19] considered infinite strings containing many distinct Ab-squares. A string of length n may contain $\Theta(n^2)$ Ab-square factors that are distinct as strings, but contains only $\mathcal{O}(n^{11/6})$ Ab-squares which are pairwise Abelian nonequivalent (correspond to different Parikh vectors), see [28]. It is also conjectured that a binary string of length n must have at least $\lfloor n/4 \rfloor$ distinct [20] and nonequivalent [21] Ab-square factors. For more conjectures related to combinatorics of Ab-square factors of strings and circular strings, see [39].

Several algorithms computing Ab-square factors of a string are known. All Ab-squares in a string of length n can be computed in $\mathcal{O}(n^2)$ time [13]. For a string over a constant-sized alphabet, all Ab-square factors of a string can be computed in $\mathcal{O}(n^2/\log^2 n + \text{output})$ time and the longest Ab-square can be computed in $\mathcal{O}(n^2/\log^2 n)$ time [29, 30]. Moreover, for a string of length n that is given by its run-length encoding consisting of r runs, the longest Ab-square that ends at each position can be computed in $\mathcal{O}(|\Sigma|(r^2 + n))$ time [2] or in $\mathcal{O}(rn)$ time [40]; both approaches require $\Omega(n^2)$ time in the worst case.

In [37] a different problem of enumerating strings being Ab-squares was considered.

Additive squares. An additive square is an even-length string over an integer alphabet such that the sums of characters of the halves of this string are the same.

► **Example 2.** The following string has exactly 4 additive squares of length 10, as shown.

1 2 0 3 2 1 2 0 2 3 2 1 0 1 2 3

All of them except for the rightmost one are also Ab-squares. This string does not contain any longer additive square. Altogether this string has 8 additive square factors.

An Ab-square (over an integer alphabet) is an additive square, but not necessarily the other way around. Combinatorially, problems related to additive squares are hard, in particular avoiding additive squares seems more difficult than avoiding Ab-squares. There are infinitely many strings over $\{0, 1, 2, 3\}$ avoiding Ab-squares, but there are only finitely many strings over the same alphabet avoiding additive squares; see [22].

In fact it is unknown if there are infinitely many strings over any finite integer alphabet avoiding additive squares [7, 25, 33]. For additive cubes the property was proved in [9] (see also [32]) however.

Nowadays, combinatorial study of Ab-square and additive square factors often involves computer experiments; see e.g. [9, 19, 36]. In addition to other applications, efficient algorithms detecting such types of squares could provide a significant aid in this research. In case of classic square factors (i.e., factors of the form XX), a linear-time algorithm for computing them is known for a string over a constant [24] and over an integer alphabet [4, 12]. We show that, unfortunately, in many cases the existence of near-linear-time algorithms for detecting Ab-square and additive square factors is unlikely, based on conjectured hardness of the 3SUM problem.

3SUM problem. The problem asks if there are distinct elements $a, b, c \in S$ such that $a + b = c$ for a given set S of n integers; see [35]. It is a general belief that the following conjecture is true for the word-RAM model.

3SUM conjecture. There is no $\mathcal{O}(n^{2-\epsilon})$ time algorithm for the 3SUM problem, for any constant $\epsilon > 0$.

A problem with input of size n is called 3SUM-hard if an $\mathcal{O}(n^{2-\epsilon})$ -time solution to the problem implies an $\mathcal{O}(n^{2-\epsilon'})$ -time solution for 3SUM, for some constants $\epsilon, \epsilon' > 0$.

Our results.

- We show that the problems of computing all centers of Ab-square factors and detecting an odd half-length Ab-square factor, called an *odd Ab-square* (consequently also computing all lengths of Ab-square factors), for a length- n string over an alphabet of size $\omega(1)$, cannot be solved in $\mathcal{O}(n^{2-\epsilon})$ time, for constant $\epsilon > 0$, unless the 3SUM conjecture fails. Weaker conditional lower bounds are also stated in the case of a constant-sized alphabet.
- For constant-sized alphabets, we show strongly sub-quadratic algorithms for these problems based on an involved result of [11] related to jumbled indexing.
- En route we prove that detection of a double 3-term arithmetic progression (see [8]) and additive squares in a length- n sequence of integers of magnitude $n^{\mathcal{O}(1)}$ is 3SUM-hard.

We obtain deterministic conditional lower bounds from a convolution version of 3SUM that is well-known to be 3SUM-hard.

Related work. In the jumbled indexing problem, we are given a text T and are to answer queries for a pattern specified by a Parikh vector which gives, for each letter of the alphabet, the number of occurrences of this letter in the pattern. For each query, we are to check if there is a factor of the text that is Ab-equivalent to the pattern (existence query) or report all such factors (reporting query). Chan and Lewenstein [11] showed a data structure that can be constructed in truly subquadratic expected time and answers existence queries in truly sublinear time for a constant-sized alphabet (deterministic constructions for very small alphabets were also shown). Amir et al. [3] showed under a 3SUM-hardness assumption that jumbled indexing with existence queries requires $\Omega(n^{2-\epsilon})$ preprocessing time or $\Omega(n^{1-\delta})$

queries for any $\epsilon, \delta > 0$ for an alphabet of size $\omega(1)$. They also provided particular constants $\epsilon_\sigma, \delta_\sigma$ for an alphabet of a constant size $\sigma \geq 3$ such that, under a stronger 3SUM-hardness assumption, jumbled indexing requires $\Omega(n^{2-\epsilon_\sigma})$ preprocessing time or $\Omega(n^{1-\delta_\sigma})$ queries. We use the techniques from both results in our algorithm and conditional lower bound for Ab-squares, respectively. The lower bound of Amir et al. was later improved and extended to both existence and reporting variants and any constant $\sigma \geq 2$ by Goldstein et al. [23, Section 7] with the aid of randomization. Moreover, recently an unconditional lower bound for the reporting variant was given in [1].

Our techniques. A subsequence of three distinct positions is a 3-term double arithmetic progression (3dap in short) if it is an arithmetic progression and the elements on these positions also form an arithmetic progression. The problem of finding a 3dap in a sequence is denoted by 3DAP. It is an odd 3dap if the first and the third positions are odd and the middle position is even. The corresponding problem is denoted by ODD-3DAP. First we reduce the convolution problem 3SUM (known to be 3SUM-hard) to the 3DAP problem via ODD-3DAP as an intermediate problem. This uses a divide-and-conquer approach and a partition of sets into sets avoiding *bad* arithmetic progression of length 3.

The 3DAP problem reduces in a simple way to detection of an additive square, showing that the latter problem is 3SUM hard.

Next, the 3DAP problem is encoded as a string. We follow the high-level idea from Amir et al. Instead of checking equality of numbers, we can check equality of their remainders modulo sufficiently many prime numbers. Then, each prime number corresponds to a distinct character. If the numbers are poly n then only $\mathcal{O}(\log n)$ prime numbers are needed. However, there is a certain technical complication, already present in the paper of Amir et al., which needs an introduction of additional gadgets working as *equalizers*. The details, compared with construction of Amir et al., are different, mostly because in the end we want to ask about detection, not indexing.

Then we consider the problem of computing all centers of Ab-squares, this requires new gadgets. We show that computing all centers of Ab-squares is 3SUM-hard, as well as detection of any Ab-square which is *well centred*.

Later we extend this to detection of any odd Ab-square. We use a construction of a string over the alphabet of size 4 with no Ab-square. The input string is “shuffled” with such a string, with some separators added. This forces odd Ab-squares to be *well centered*, in this way we reduce the previously considered problem of detection of any well-centred Ab-square to the detection of any odd Ab-square. Ultimately, this shows that the latter problem is 3SUM-hard.

2 From Conv3SUM to finding double 3-term arithmetic progressions

For integers a, b , by $[a, b]$ we denote the set $\{a, \dots, b\}$. We use the following convolution variant of the 3SUM problem that is 3SUM-hard; see [10, 31, 35] for both randomized and deterministic reductions. As already noted in [3], the range of elements can be made $[-N^2, N^2]$ using a randomized hashing reduction from [5, 35].

CONV3SUM(\bar{x})

Input: A sequence $\bar{x} = [x_1, \dots, x_N] \in [-N^2, N^2]$

Output: Yes if there are $i \neq j$ such that $x_i + x_j = x_{i+j}$; no otherwise.

Let us denote $\text{mid}(a, b) = (a + b)/2$ and define the condition:

$$\Lambda_{\bar{x}}(i, j) = (i \neq j \wedge j - i \text{ is even} \wedge x_{\text{mid}(i, j)} = \text{mid}(x_i, x_j)).$$

We omit the subscript \bar{x} if it is clear from the context. The last part of the condition is equivalent to $x_j - x_{\text{mid}(i, j)} = x_{\text{mid}(i, j)} - x_i$.

Our first goal is to reduce the CONV3SUM problem to the following one with $K = N^{\mathcal{O}(1)}$.

Double 3-Term Arithmetic Progression, 3DAP(\bar{x})

Input: $\bar{x} = [x_1, \dots, x_n]$, each of x_i is in $[0, K]$.

Output: $(\exists i, j) \Lambda(i, j)$.

In Section 2.1 we obtain a reduction of CONV3SUM to an intermediate version of 3DAP with additional constraints on i, j , and in Section 2.2 we show how these constraints can be avoided.

2.1 From Conv3SUM to Odd-3DAP

Let us fix an integer sequence x_1, \dots, x_N . For an arithmetic progression (arithmetic sequence) $\mathcal{I} = i_1, \dots, i_n$, where $1 \leq i_1 < \dots < i_n \leq N$, i.e. $i_2 - i_1 = \dots = i_n - i_{n-1}$, we define the following extended functions.

$$\text{CONV3SUM}(\bar{x}, \mathcal{I}) = (\exists i_a, i_b \in \mathcal{I} : x_{i_a} + x_{i_b} = x_{i_a + i_b}, i_a < i_b)$$

$$\text{ODDCONV3SUM}(\bar{x}, \mathcal{I}) = (\exists i_a, i_b \in \mathcal{I} : x_{i_a} + x_{i_b} = x_{i_a + i_b}, a + b \text{ is odd}).$$

Note that it can happen that $i_a + i_b \notin \mathcal{I}$. For a fixed \bar{x} the input size is $|\mathcal{I}|$.

► **Lemma 3.** *An instance of $\text{CONV3SUM}(\bar{x})$ can be reduced to an alternative of $\mathcal{O}(N)$ instances of $\text{ODDCONV3SUM}(\bar{x}, \mathcal{I})$ of total size $\mathcal{O}(N \log N)$ in $\mathcal{O}(N \log N)$ time.*

Proof. If $\mathcal{I} = i_1, \dots, i_n$, by \mathcal{I}_{odd} and $\mathcal{I}_{\text{even}}$ we denote the subsequences i_1, i_3, \dots and i_2, i_4, \dots , respectively. We proceed recursively as shown in the following function CONV3SUM, with the first call to $\text{CONV3SUM}(\bar{x}, [1, 2, \dots, N])$.

function CONV3SUM(\bar{x}, \mathcal{I})

Comment: \mathcal{I} is an arithmetic progression

if $|\mathcal{I}| \leq 2$ **then return false;**

return ODDCONV3SUM(\bar{x}, \mathcal{I}) \vee CONV3SUM($\bar{x}, \mathcal{I}_{\text{odd}}$) \vee CONV3SUM($\bar{x}, \mathcal{I}_{\text{even}}$);

Correctness. Let $\mathcal{I} = i_1, \dots, i_n$ and assume there are two indices a, b such that $x_{i_a} + x_{i_b} = x_{i_a + i_b}$. If $a + b$ is odd, then ODDCONV3SUM(\bar{x}, \mathcal{I}) returns true. Otherwise both a, b are of the same parity, so $i_a, i_b \in \mathcal{I}_{\text{odd}}$ or $i_a, i_b \in \mathcal{I}_{\text{even}}$. Consequently, the problem is split recursively into subproblems that correspond to \mathcal{I}_{odd} and $\mathcal{I}_{\text{even}}$.

Complexity. Let us observe that one call to CONV3SUM(\bar{x}, \mathcal{I}) creates an instance of ODDCONV3SUM of $\mathcal{O}(|\mathcal{I}|)$ size in $\mathcal{O}(|\mathcal{I}|)$ time (\bar{x} does not change). Let $\#(n)$ and $S(n)$ denote the total number and size of all instances of \mathcal{I} generated by CONV3SUM(\bar{x}, \mathcal{I}), when initially $|\mathcal{I}| = n$. We then have

$$\#(n) = \#(\lfloor n/2 \rfloor) + \#(\lceil n/2 \rceil) + 1 \quad \text{and} \quad S(n) = S(\lfloor n/2 \rfloor) + S(\lceil n/2 \rceil) + \Theta(n),$$

which yields $\#(N) = \mathcal{O}(N)$ and $S(N) = \mathcal{O}(N \log N)$. The reduction takes $\mathcal{O}(S(N))$ time. ◀

77:6 Hardness of Detecting Abelian and Additive Squares

We say that a 3-element arithmetic progression is a *good* progression if the middle element is even and two others are odd and introduce the following problem.

ODD-3DAP(\bar{x})

Input: $\bar{x} = [x_1, \dots, x_n]$, each of x_i is in $[-\mathcal{O}(N^2), \mathcal{O}(N^2)]$.

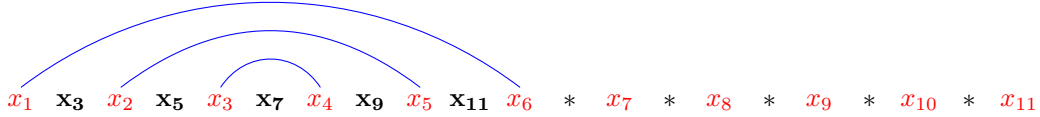
Output: $(\exists i, j) [\mathbf{A}(i, j) \text{ and } (i, \text{mid}(i, j), j) \text{ is a good progression}]$.

► **Lemma 4.** $\text{ODD CONV 3SUM}(\bar{x}, \mathcal{I})$ is reducible in $\mathcal{O}(|\mathcal{I}|)$ time and space to $\text{ODD-3DAP}(\bar{y})$, where $|\bar{y}| = \mathcal{O}(|\mathcal{I}|)$.

Proof. Let $\mathcal{I} = i_1, \dots, i_n$. Define $\alpha_N = 2N^2 + 1$ and let \bar{y} be a sequence of length $2n - 1$ that is created as follows:

1. put $x_{i_1}, x_{i_2}, \dots, x_{i_n}$ at subsequent odd positions in \bar{y} ;
2. at each even position $2j$, put $x_{i_j + i_{j+1}}$ or, if $i_j + i_{j+1} > N$, put α_N .
3. multiply elements on even positions by 2.

After the first two steps $\text{ODD CONV 3SUM}(\bar{x}, \mathcal{I})$ is equivalent to $(\exists i, j) y_{\text{mid}(i, j)} = y_i + y_j$ for odd i, j and even $\text{mid}(i, j)$; see Figure 1. Then, after the third step, $\text{ODD CONV 3SUM}(\bar{x}, \mathcal{I})$ is equivalent to $\text{ODD-3DAP}(\bar{y})$. ◀



■ **Figure 1** The sequence constructed in Lemma 4 for $\bar{x} = [x_1, x_2, x_3, \dots, x_{11}]$ and $\mathcal{I} = (1, 2, \dots, 11)$ after the first two steps (* denotes α_{11}). Note that the elements connected by arcs all have their sum of indices equal to 7; this is because \mathcal{I} is an arithmetic progression.

2.2 From Odd-3DAP to 3DAP

Our main tool in this subsection is partitioning a set of integers into progression-free sets. A set of integers A is called *progression-free* if it does not contain a non-constant three-element arithmetic progression. We use the following recent result that extends a classical paper of Behrend [6].

► **Theorem 5** ([14]). Any set $A \subseteq [1, n]$ can be partitioned into $n^{o(1)}$ progression-free sets in $n^{1+o(1)}$ time.

► **Lemma 6.** We can construct in $n^{1+o(1)}$ time a family \mathcal{F} of $n^{o(1)}$ subsets of $[1, n]$ satisfying:

- (a) Each good 3-element progression is contained in some $S \in \mathcal{F}$.
- (b) If $S \in \mathcal{F}$, then all 3-element arithmetic progressions in S are good.

Proof. Let us divide the elements from $[1, n]$ into three classes:

$$\text{BLUE} = \{i \leq n : i \text{ is even}\},$$

$$\text{RED} = \{i \leq n : i \bmod 4 = 1\}, \quad \text{GREEN} = \{i \leq n : i \bmod 4 = 3\}.$$

Each element $i \in [1, n]$ has the colour blue, red or green of its corresponding class. Each class forms an arithmetic progression.

A progression is called *multi-chromatic* if its elements are of three distinct colours. Let us observe that a 3-element progression is good if and only if it is multi-chromatic. Indeed, this is because if $i, j \in \text{RED}$ (or GREEN), then $\text{mid}(i, j)$ is odd.

Now instead of good progressions we will deal with multi-chromatic progressions. We treat sets of integers as increasing sequences and for a set $C = \{c_1, \dots, c_m\}$ we denote by C_{odd} and C_{even} the subsets $\{c_1, c_3, \dots\}$ and $\{c_2, c_4, \dots\}$.

For example $\text{BLUE}_{\text{odd}} = \{i \leq n : i \bmod 4 = 2\}$, $\text{RED}_{\text{even}} = \{i \leq n : i \bmod 8 = 5\}$.

Our construction works as follows:

1. Partition the set $[1, n]$ into classes $\text{BLUE}, \text{RED}, \text{GREEN}$.
2. For each class $C \in \{\text{BLUE}, \text{RED}, \text{GREEN}\}$ partition it in $n^{1+o(1)}$ time into a family \mathcal{F}_C of $n^{o(1)}$ progression-free sets with the use of Theorem 5.
3. Refine each partition \mathcal{F}_C , splitting each set $X \in \mathcal{F}_C$ into two sets $X \cap C_{\text{odd}}$, $X \cap C_{\text{even}}$, so that for each set X in the new refined partition \mathcal{F}_C we have $X \subseteq C_{\text{odd}}$ or $X \subseteq C_{\text{even}}$. Each family is still of size $n^{o(1)}$.
4. Return $\mathcal{F} = \{X \cup Y \cup Z : X \in \mathcal{F}_{\text{BLUE}}, Y \in \mathcal{F}_{\text{RED}}, Z \in \mathcal{F}_{\text{GREEN}}\}$.

Proof of point (a). Each multi-chromatic progression is contained in some $S \in \mathcal{F}$ since each element of C is contained in a set from \mathcal{F}_C .

Proof of point (b). The proof is by contradiction. Assume that $S \in \mathcal{F}$ contains a progression which is not multi-chromatic. There are two cases.

Case 1: the progression is monochromatic, hence it appears in a single set $X \in \mathcal{F}_C$. However every X is progression-free (step 2), hence such a progression cannot appear in any $S \in \mathcal{F}$; a contradiction.

Case 2: the progression contains exactly two different colors. Observe that if $i \bmod p = \text{mid}(i, j) \bmod p = r$, then $j \bmod p = r$ (if the middle element of progression belongs to the same class as one of the other elements, then the triple is monochromatic), hence the two-coloured arithmetic progression has to consist of $i, j \in C$ and $\text{mid}(i, j) \notin C$.

Since i, j both belong to C_{odd} or C_{even} (step 3), $\text{mid}(i, j)$ must belong to C (if $i \bmod 2p = j \bmod 2p$, then $i \bmod p = \text{mid}(i, j) \bmod p$). Consequently, the progression cannot contain exactly two colours; a contradiction. \blacktriangleleft

Our next tool is a *deactivation* of a set of elements which indexes are not in a given set E , that is, omitting them in the computation of a solution. For $E \subseteq [1, n]$ the operation $\text{restr}(\bar{x}, E)$ replaces each element x_i on position $i \notin E$ by $5 \max\{MAX, n^2\} + i^2$, where $MAX = \max_{k \geq 1} |x_k|$.

► **Lemma 7.** $3\text{DAP}(\text{restr}(\bar{x}, E)) \iff (\exists i, j) \Lambda_{\bar{x}}(i, j) \wedge i, j, \text{mid}(i, j) \in E$.

Proof. The (\Leftarrow) part is obvious, so it suffices to show (\Rightarrow) . If at least one, but not all, of $i, j, \text{mid}(i, j)$ is not in E , then $\Lambda_{\bar{y}}(i, j)$ cannot hold for $\bar{y} = \text{restr}(\bar{x}, E)$ because $y_{\text{mid}(i, j)}$ and $\text{mid}(y_i, y_j)$ differ by at least $\max\{\max_k \{|x_k|\}, n^2\}$ (for an exhaustive check of all the cases, see the full version). Otherwise, if all the positions $i, j, \text{mid}(i, j)$ are not in E , then $\Lambda_{\bar{y}}(i, j)$ does not hold because $\text{mid}(i^2, j^2) - (\frac{i+j}{2})^2 = (\frac{i-j}{2})^2 > 0$ since $i \neq j$. \blacktriangleleft

An instance \bar{x} is called an **odd-half** instance if $\Lambda(i, j)$ is false for i, j such that $(j - i)/2$ is even (equivalently, for i, j such that i and $\text{mid}(i, j)$ have the same parity). Efficient equivalence

$$\text{ODD-3DAP}(\bar{x}) \iff (\exists S \in \mathcal{F}) 3\text{DAP}(\text{restr}(\bar{x}, S))$$

follows now from Lemmas 6 and 7.

This produces only odd-half instances because only good progressions are left in the construction of Lemma 6. The instances have elements in $[-\mathcal{O}(N^2), \mathcal{O}(N^2)]$. We can increase all the elements by $\mathcal{O}(N^2)$ so that they become non-negative. This implies:

► **Lemma 8.** *An instance of ODD-3DAP can be reduced in $n^{1+o(1)}$ time to $n^{o(1)}$ odd-half instances of 3DAP of total size $n^{1+o(1)}$ and with elements up to $K = \mathcal{O}(N^2)$.*

Finally, we show that the resulting instances can be glued together to a single equivalent one.

► **Theorem 9.** *An instance of CONV3SUM can be reduced in $N^{1+o(1)}$ time to an odd-half instance of 3DAP of size $n = N^{1+o(1)}$ with elements up to $K = N^{3+o(1)}$.*

Proof. With Lemmas 3, 4, and 8 we obtain a reduction from CONV3SUM to $N^{1+o(1)}$ odd-half instances of 3DAP of total size $N^{1+o(1)}$. The instances have elements in $[0, \mathcal{O}(N^2)]$. We will show that these instances can be reduced to a single odd-half instance of 3DAP of size $N^{1+o(1)}$ with elements in the range $[0, N^{3+o(1)}]$ in time $N^{1+o(1)}$. The resulting instance will return true if and only if at least one of the input instances does.

Let $t = N^{1+o(1)}$ be the number of the instances of 3DAP, numbered 1 through t . We use Theorem 5¹ and pick the largest constructed progression-free set $A \subseteq [1, m]$, for some m . By the pigeonhole principle, $|A| \geq m^{1-o(1)}$. We select m that is large enough so that $m^{1-o(1)} \geq t$, so $m = t^{1+o(1)} = N^{1+o(1)}$, and trim the set A to the size t . Let $A = \{a_1, \dots, a_t\}$. For instance i we multiply all its elements by $2m$ and add to each element the value a_i . Finally we concatenate all the instances.

If any of the input instances returns true, then so does the output instance, since multiplication by and addition of the same number to all elements cannot affect the outcome of a single instance. If none of the input instances returns true, then the only possibility for the output instance to return true is to contain a 3-element arithmetic progression with elements from multiple parts corresponding to the input instances. However, this is impossible since, taken modulo $2m$, the progression would form an arithmetic progression in the set A . ◀

► **Corollary 10.** *The general 3DAP problem is also 3SUM-hard.*

► **Remark 11.** Similarly as in CONV3SUM, techniques from [5] can be used to hash down the range in 3DAP to integers of magnitude $\mathcal{O}(N^2)$ (cf. [3]), using randomization.

► **Remark 12.** The AVERAGE problem (introduced by J. Erickson [17]) asks if there are distinct elements $a, b, c \in S$ such that $a + b = 2c$ for a given set S of n integers. It was recently shown to be 3SUM-hard [14]. The 3DAP problem can be viewed as a convolution version of the AVERAGE problem². The ideas based on almost linear hashing used in the reductions from 3SUM to CONV3SUM [35, 10] can be extended with some effort to reduce AVERAGE to 3DAP. We presented a different reduction that additionally directly leads to an instance of 3DAP with an odd-half property, which is essential in our proof of 3SUM-hardness of computing Ab-squares (see the proof of Lemma 19).

¹ Actually, a deterministic version of Behrend's construction from [14] or an earlier construction of Salem and Spencer [38] would suffice here.

² <https://cs.stackexchange.com/questions/10681/is-detecting-doubly-arithmetic-progressions-3sum-hard/10725#10725>

2.3 Hardness of detecting additive squares

If the alphabet is a set of integers, then a string W is called an *additive square* if $W = UV$, where $|U| = |V|$ and $\sum_{i=1}^{|U|} U[i] = \sum_{i=1}^{|V|} V[i]$.

► **Theorem 13.** *Finding an additive square in a length- N sequence composed of integers of magnitude $N^{O(1)}$ is 3SUM-hard.*

Proof. We use Theorem 9 to reduce CONV3SUM to an instance of 3DAP of size $n = N^{1+o(1)}$ with elements in the requested range. 3DAP returns true on an instance x_1, \dots, x_n if and only if the sequence $x_2 - x_1, x_3 - x_2, \dots, x_n - x_{n-1}$ contains an additive square. As the reduction works in $N^{1+o(1)}$ total time, the conclusion follows. ◀

3 From arithmetics to Abelian stringology

We use capital letters to denote strings and lower case Greek letters to denote sets of integers. We assume that the positions in a string S are numbered 1 through $|S|$, where $|S|$ denotes the length of S . By $S[i]$ and $S[i..j]$ we denote the i th letter of S and the string $S[i] \cdots S[j]$ called a factor of S . The reverse of string S , i.e. the string $S[|S|] \cdots S[1]$, is denoted as S^R . By ε we denote the empty string. By $\text{Alph}(S)$ we denote the set of distinct letters in S .

We denote Ab-equivalence of U and V by $U \cong V$. For a string U , by $\text{Parikh}(U)$ we denote the Parikh vector of U . Then $U \cong V$ if and only if $\text{Alph}(U) = \text{Alph}(V)$ and $\text{Parikh}(U) = \text{Parikh}(V)$.

We use an encoding of Amir et al. [3] based on the Chinese remainder theorem to connect CONV3SUM-type problems with Abelian stringology.

Let $p_1 < p_2 < \dots < p_k$ be prime numbers. The Chinese remainder theorem states that if one knows the remainders r_1, r_2, \dots, r_k of an integer x , such that $0 \leq x < \prod p_i$, when dividing by p_i 's, then one can uniquely determine x . Assuming that the remainders of an integer x are r_1, r_2, \dots, r_k , we could encode x as a possibly short string $\mathbf{a}_1^{r_1} \mathbf{a}_2^{r_2} \cdots \mathbf{a}_k^{r_k}$ over an alphabet $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k\}$ (the symbols correspond to consecutive prime numbers).

For example for primes 2,3,5 the encoding of 11 would be $\mathbf{a}_1^1 \mathbf{a}_2^2 \mathbf{a}_3^1$ since its remainders modulo 2,3,5 are 1,2,1, respectively. However, we are interested in encodings of subtractions of one number from another one, and it is more complicated.

Let $\bar{x} = [x_1, \dots, x_n]$ be an instance of 3DAP and $r_1^{(i)}, r_2^{(i)}, \dots, r_k^{(i)}$ be remainders of x_i modulo p_1, p_2, \dots, p_k . Like Amir et al. [3], we define for $1 \leq i < n$ and $1 \leq j \leq k$,

$$\text{EXP}_i(j) = r_j^{(i+1)} - r_j^{(i)} + d \text{ where } d = \max_{j=1}^k p_j, \quad \mathbf{S}_i = \mathbf{a}_1^{\text{EXP}_i(1)} \mathbf{a}_2^{\text{EXP}_i(2)} \cdots \mathbf{a}_k^{\text{EXP}_i(k)}.$$

We choose a sequence p_1, \dots, p_k of k distinct primes such that $p_1 \cdots p_k > \max\{x_i\}$. In this way we encode the difference $x_j - x_i$, for $j > i$, by a string $\mathbf{S}_i \mathbf{S}_{i+1} \cdots \mathbf{S}_{j-1}$. An obstacle is the potentially possible inequality $(a \bmod p) - (b \bmod p) \neq (a - b) \bmod p$. However a small correction is sufficient, due to the following observation.

► **Observation 14.** $(a \bmod p) - (b \bmod p) + q = (a - b) \bmod p$, where $q \in \{0, p\}$.

If we apply the encoding to an instance $\bar{x} = x_1, \dots, x_n$ of 3DAP, we obtain a lemma that is analogous to [3, Lemma 1].

► **Lemma 15.** $\Lambda(i, j)$ holds for $i < j$, $j-i$ even, iff for each $t \in [1, k]$, there are $e_t, f_t \in \{0, p_t\}$, such that

$$e_t + \text{EXP}_i(t) + \text{EXP}_{i+1}(t) + \cdots + \text{EXP}_{\text{mid}(i,j)-1}(t) = \\ \text{EXP}_{\text{mid}(i,j)}(t) + \text{EXP}_{\text{mid}(i,j)+1}(t) + \cdots + \text{EXP}_{j-1}(t) + f_t.$$

Let Ψ be a morphism such that $\Psi(i) = \mathbf{a}_i^{p_i}$ for each $i = 1, \dots, k$. We treat a set $U = \{u_1, \dots, u_w\}$ as a string $u_1 \cdots u_w$, where $u_1 < u_2 < \dots < u_w$. If we interpret the vector $(EXP_i(1), EXP_i(2), \dots, EXP_i(k))$ as \mathbf{S}_i , then Lemma 15 directly implies the following fact.

► **Lemma 16.** *Assume $i < j$ and $j - i$ is even. Then*

$$\Lambda(i, j) \iff (\Psi(\alpha) \mathbf{S}_i \mathbf{S}_{i+1} \cdots \mathbf{S}_{mid(i,j)-1} \cong \mathbf{S}_{mid(i,j)} \cdots \mathbf{S}_{j-1} \Psi(\beta))$$

for some disjoint subsets α, β of $[1, k]$.

4 Hardness of computing all centers of Ab-squares

We construct a text T over the alphabet $\{\mathbf{a}_1, \dots, \mathbf{a}_k, \mathbf{b}, \bullet, \star, \#, \$\}$ such that 3DAP has a solution if and only if T contains an Ab-square with one of specified centers, so-called *well-placed* Ab-square.

First we extend each \mathbf{S}_i to have the same length $M \geq \max_{i=1}^{n-1} |\mathbf{S}_i|$, to be defined later. Intuitively, it is needed to control the number of \mathbf{S}_i 's in the strings from Lemma 16. We append $M - |\mathbf{S}_i|$ occurrences of a letter \mathbf{b} to each \mathbf{S}_i . Let \mathbf{S}_i^I denote this modified string.

Lemma 16 immediately implies the following fact.

► **Lemma 17.** *Assume $i < j$ and $j - i$ is even. Then*

$$\Lambda(i, j) \iff (\mathbf{b}^e \Psi(\alpha) \mathbf{S}_i^I \mathbf{S}_{i+1}^I \cdots \mathbf{S}_{mid(i,j)-1}^I \cong \mathbf{S}_{mid(i,j)}^I \cdots \mathbf{S}_{j-1}^I \Psi(\beta) \mathbf{b}^f),$$

for some disjoint subsets α, β of $[1, k]$, where $e + |\Psi(\alpha)| = f + |\Psi(\beta)|$ with $\min(e, f) = 0$.

The parts $\mathbf{b}^e \Psi(\alpha)$, $\Psi(\beta) \mathbf{b}^f$ in the above lemma can be treated as *equalizers*. Let us note that in the above lemma we can assume that $\max(e, f) \leq \max(|\Psi(\alpha)|, |\Psi(\beta)|) \leq kd$.

A pair of disjoint sets α, β that satisfies $\alpha \cup \beta = [1, k]$ will be called a *2-partition* of $[1, k]$. For a 2-partition (α, β) of $[1, k]$, we use the string

$$\Gamma(\alpha, \beta) = \# \alpha \$ \mathbf{b}^{kd} \# \beta \$,$$

called a Γ -string. If $k = 4$, $d = 7$, an example of a Γ -string is $\Gamma(2, 134) = \# 2 \$ \mathbf{b}^{28} \# 134 \$$.

Let $(\pi_1, \pi'_1), (\pi_2, \pi'_2), \dots, (\pi_m, \pi'_m)$ be the sequence of all $m = 4^k$ pairs of Γ -strings. Define

$$U = \pi_m \pi_{m-1} \dots \pi_1, \quad V = \pi'_1 \pi'_2 \dots \pi'_m.$$

We have $\{\pi_1, \dots, \pi_m\} = \{\pi'_1, \dots, \pi'_m\}$, so $U \cong V$.

► **Observation 18.** *For disjoint subsets $\alpha, \beta \subseteq [1, k]$ and integers $0 \leq e, f \leq kd$, there are decompositions $U = U_1 \mathbf{b}^e \# \beta \$ U_2$ and $V = V_1 \# \alpha \$ \mathbf{b}^f V_2$, where $U_2 \cong V_1$.*

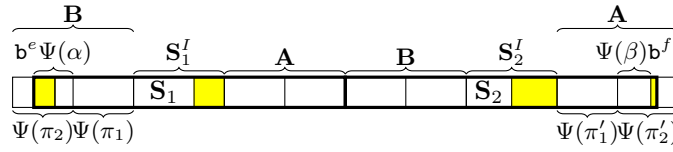
Let us recall the morphism Ψ such that $\Psi(i) = \mathbf{a}_i^{p_i}$ for each $i \in [1, k]$. We define additionally $\Psi(c) = c$ for $c \in \{\mathbf{b}, \#, \$\}$ and set

$$\mathbf{B} = \Psi(U), \quad \mathbf{A} = \Psi(V), \quad M = |\mathbf{A}| = |\mathbf{B}|.$$

Let us observe that indeed $\max_{i=1}^{n-1} |\mathbf{S}_i| \leq M$ holds since $|\mathbf{S}_i| \leq kd + \sum_{j=1}^k p_j$ and the length of $\Psi(W)$ for any Γ -string W is $kd + \sum_{j=1}^k p_j + 4$.

We add two new letters \bullet, \star and define the following string (the symbols “ \downarrow ” are not parts of the string, but only show supposed centers of Ab-squares).

$$T = \bullet \mathbf{B} \star \mathbf{S}_1^I \mathbf{A} \bullet \overset{\text{center}}{\downarrow} \star \mathbf{B} \bullet \mathbf{S}_2^I \mathbf{A} \star \overset{\text{center}}{\downarrow} \bullet \mathbf{B} \star \mathbf{S}_3^I \mathbf{A} \bullet \overset{\text{center}}{\downarrow} \star \mathbf{B} \bullet \mathbf{S}_4^I \mathbf{A} \star \dots \quad (1)$$



■ **Figure 2** Internal structure of an Ab-square, shown in a thick box (proportions are symbolic), in $\mathbf{BS}_1^I \mathbf{A} \mathbf{BS}_2^I \mathbf{A}$. Here $x_{mid(1,3)} = mid(x_1, x_3)$ and $\mathbf{b}^e \Psi(\alpha)$, $\Psi(\beta) \mathbf{b}^f$ are equalizers.

An Ab-square is called *well-placed* if its center is between the letters \bullet, \star in any order. Recall that, due to Theorem 9, we can assume that the input to 3DAP guarantees that only odd-half instances could have solutions.

► **Lemma 19.** *Assume \bar{x} is an odd-half instance. Then $3DAP(\bar{x})$ has a solution if and only if T contains a well-placed Ab-square.*

Proof. Let $\bar{x} = [x_1, \dots, x_n]$ be an odd-half instance of 3DAP. We show two implications.

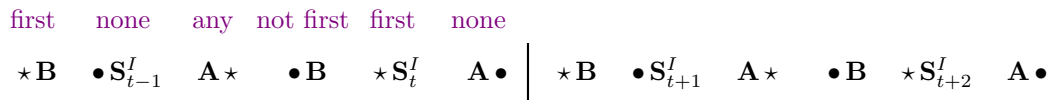
(\Rightarrow) Assume that $\mathbf{A}(i, j)$ holds for \bar{x} . Lemma 17 implies that for strings W, Z such that $W \cong Z$ we have

$$\begin{aligned} \mathbf{b}^e \# \Psi(\alpha) \$ W \star \mathbf{S}_i^I \mathbf{A} \bullet \star \mathbf{B} \bullet \mathbf{S}_{i+1}^I \mathbf{A} \star \dots \bullet \mathbf{B} \star \mathbf{S}_{mid(i,j)-1}^I \mathbf{A} \bullet \cong \\ \star \mathbf{B} \bullet \mathbf{S}_{mid(i,j)}^I \mathbf{A} \star \dots \bullet \mathbf{B} \star \mathbf{S}_{j-2}^I \mathbf{A} \bullet \star \mathbf{B} \bullet \mathbf{S}_{j-1}^I \mathbf{A} \bullet \# \Psi(\beta) \$ \mathbf{b}^f \end{aligned} \quad (2)$$

for some disjoint subsets α, β of $[1, k]$, where $e + |\Psi(\alpha)| = f + |\Psi(\beta)|$ with $\min(e, f) = 0$. Indeed, we use the fact that $\mathbf{A} \cong \mathbf{B}$ and the counts of letters \bullet and \star on both hand sides are equal (because $(j - i)/2$ is odd). By Observation 18, we obtain a well-placed Ab-square in T (or we obtain it after exchanging all letters \bullet with \star).

(\Leftarrow) Assume that T has a well-placed Ab-square factor with center immediately after $\bullet \mathbf{B} \star \mathbf{S}_t^I \mathbf{A} \bullet$ (the case that it is immediately after $\star \mathbf{B} \bullet \mathbf{S}_t^I \mathbf{A} \star$ is symmetric). Let us investigate what can be the position s of the first letter of this Ab-square.

Recall that $|\mathbf{S}_i^I| = |\mathbf{A}| = |\mathbf{B}| = M$ for each $i \in [1, n - 1]$, so T can be seen as composed of blocks of length $M' = M + 1$. We will check which of these blocks can contain s , by checking the counts of each of the letters \bullet, \star in both halves of the Ab-square. The positions of letters \bullet, \star in T repeat with period $6(M + 1)$, so it is sufficient to inspect the first 6 blocks on each side, as the remaining ones will behave periodically; see Figures 3 and 4.



■ **Figure 3** Which position in a block of T can be the starting position of a well-placed Ab-square with the designated center, just counting letters \bullet, \star .

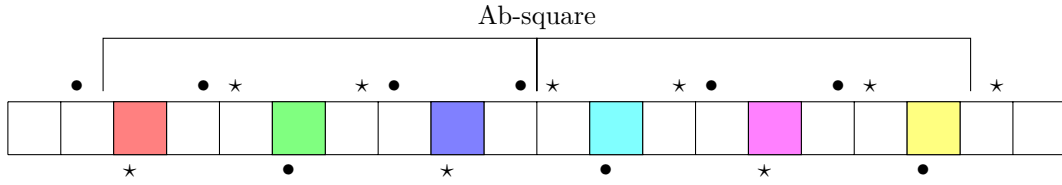
By counting letters \bullet, \star in both halves of the Ab-square, it can be readily verified that s cannot be in any block $\mathbf{A} \bullet$ or $\bullet \mathbf{S}_i^I$; if in any block $\star \mathbf{B}$ or $\star \mathbf{S}_i^I$, it can only be the first position of the block; it cannot be the first position in a block $\bullet \mathbf{B}$; and it can be in any position in a block $\mathbf{A} \star$.

Moreover, s cannot be the first position in a block $\star \mathbf{B}$, since this would imply, by Lemma 17, that $\mathbf{A}(i, j)$ holds for i such that the block $\bullet \mathbf{S}_i^I$ immediately follows the $\star \mathbf{B}$ block and $j = 2t - i$. However, in this case $(j - i)/2$ is even, which is impossible.

If s is the first position of a block $\star S_i^I$, then this implies, again by Lemma 17, that $\Lambda(i, j)$ holds for $j = 2t - i$. In this case $(j - i)/2$ is odd, so this is a valid solution to the corresponding 3DAP instance.

We are left with the case that s belongs to a block $\mathbf{A}\star$ or $\bullet\mathbf{B}$ (and in case of $\bullet\mathbf{B}$ does not coincide with the position of the letter \bullet). Henceforth it suffices to count letters different from \bullet, \star in the halves. Each of the gadgets \mathbf{A}, \mathbf{B} is a concatenation of m Ab-equivalent strings of the form $\# \Psi(\alpha) \$ \mathbf{b}^{kd} \# \Psi(\beta) \$$, where $\Psi(\alpha), \Psi(\beta)$ are composed of letters \mathbf{a}_i only. By counting the letters $\#$ and $\$$ in both halves of the Ab-square, we see that s can only be a position which holds the letter \mathbf{b} or $\#$.

Hence, the Ab-square is necessarily of the form (2), which, by Lemma 17, implies that $\Lambda(i, j)$ holds, where $\star S_i^I$ is the first such block after the position s and $j = 2t - i$. ◀



■ **Figure 4** The global structure of a fragment containing a well-placed Ab-square; there are three types of blocks: $c\mathbf{B}$, cS_i^I , $\mathbf{A}c$, where c is one of \bullet, \star . The blocks of the second type (which can be considered as essential blocks) are in color, each block is of length $M + 1$ (recall that $|\mathbf{A}| = |\mathbf{B}| = |S_i| = M$). The special letters \bullet, \star force each half of a well-placed Ab-square to contain a number of full S_i 's.

► **Theorem 20.** *Computing all positions that are centers of Ab-square factors in a length- n string over an alphabet of size $\omega(1)$ is 3SUM-hard.*

Proof. Due to Theorem 9 we can reduce CONV3SUM in $N^{1+o(1)}$ time to an odd-half instance \bar{x} of 3DAP of size $n = N^{1+o(1)}$ with elements in the range $[0, N^{3+o(1)}]$.

We construct the string T as shown in Eq. 1 for the sequence \bar{x} . Then Lemma 19 implies that 3DAP is a YES-instance if and only if T has a well-placed Ab-square. The string T has length $\mathcal{O}(N^{1+o(1)}M)$. Each of the strings \mathbf{A}, \mathbf{B} has length M and is composed of $m = 4^k$ strings of length $\mathcal{O}(kd)$, i.e., Ψ -images of Γ -strings.

Hence, $M = \mathcal{O}(4^k kd)$. We select k such that $k = \omega(1)$ and simultaneously $k = \mathcal{O}(\log N / \log \log N)$. Then we have $4^k k = N^{o(1)}$ and the k primes are of magnitude $d = \mathcal{O}(N^{(3+o(1))/k}) = N^{o(1)}$ (we can choose k consecutive primes computed using Eratosthenes's sieve).

Overall $|T| = N^{1+o(1)}$ and $|\text{Alph}(T)| \leq k + 5 = \omega(1)$. (One can obtain any alphabet up to $\mathcal{O}(N)$ by appending distinct letters to T .) ◀

With the same argument for a constant-sized alphabet we obtain the following result.

► **Theorem 21.** *All positions that are centers of Ab-square factors in a length- n string over an alphabet of size $5 + k$, for a constant k , cannot be computed in $\mathcal{O}(n^{2 - \frac{6}{3+k} - \varepsilon})$ time, for a constant $\varepsilon > 0$, unless the 3SUM conjecture fails.*

5 Computing centers of Ab-squares for constant-sized alphabets

A set of vectors in $[1, n]^d$ is called *monotone* if its elements can be ordered so that they form a monotone non-decreasing sequence on each coordinate.

► **Definition 22.** For sets \mathcal{A} and \mathcal{B} of vectors we define $\mathcal{A} + \mathcal{B} = \{a + b : a \in \mathcal{A}, b \in \mathcal{B}\}$ and $c \cdot \mathcal{A} = \{ca : a \in \mathcal{A}\}$, and for a string W we define: $P_{l,r}(W) = \{\text{Parikh}(W[l..k]) : l \leq k \leq r\}$. Let us denote by $|A|$ the length of a string corresponding to a Parikh vector A .

In the algorithm we use the following fact shown in [11]. The exact complexities can be found in [11, Theorem 3.1].

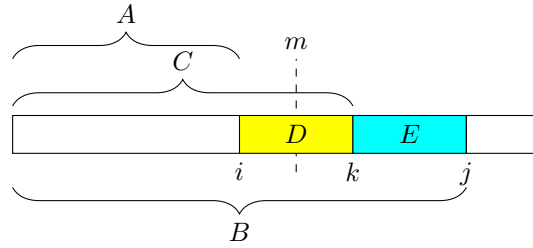
► **Fact 1** ([11]). Given three monotone sequences $\mathcal{A}, \mathcal{B}, \mathcal{C}$ in $[1, n]^d$ for a constant d , we can compute $(\mathcal{A} + \mathcal{B}) \cap \mathcal{C}$ in $\mathcal{O}(n^{2-\epsilon})$ expected time for a constant $\epsilon > 0$, or in $\mathcal{O}(n^{2-\epsilon'})$ worst case time for a constant $\epsilon' > 0$ if $d \leq 7$.

■ **Algorithm 1** CENTERS(T).

```

if  $|T| < 2$  return  $\emptyset$ ;
 $m = \lceil n/2 \rceil$ ;
 $\mathcal{A} = P_{0,m-1}(T)$ ;  $\mathcal{B} = P_{m,n}(T)$ ;  $\mathcal{C} = P_{0,n}(T)$ ;
 $\mathcal{M} = \{|C| : 2C \in (\mathcal{A} + \mathcal{B}) \cap 2 \cdot \mathcal{C}\}$ ;
 $T_{\text{left}} = T[1..m-1]$ ;  $T_{\text{right}} = T[m..n]$ ;
return  $\mathcal{M} \cup \text{CENTERS}(T_{\text{left}}) \cup \{k + m : k \in \text{CENTERS}(T_{\text{right}})\}$ 

```



■ **Figure 5** $A \in \mathcal{A}$, $B \in \mathcal{B}$, $C \in \mathcal{C}$ denote Parikh vectors of the corresponding fragments. If $A + B = 2C$, then $D = E$ and $k = |C|$ is a center of an Ab-square.

► **Theorem 23.** For a string of length n over an alphabet of size $d = \mathcal{O}(1)$, we can compute centers of all Ab-squares and centers of all odd Ab-squares in expected time $\mathcal{O}(n^{2-\epsilon})$ or in worst case time $\mathcal{O}(n^{2-\epsilon})$ if $d \leq 7$, for $\epsilon > 0$.

Proof. We use the above algorithm. Correctness of the algorithm is straightforward; see Figure 5. If

$$|A| < |B|, \quad |C| = (|A| + |B|)/2, \quad B = A + D + E, \quad C = A + D$$

then $A + B = 2C \iff 2A + D + E = 2A + 2D$.

Consequently, after cancelling the same parts on both sides, $A + B = 2C \iff E = D$, equivalently if and only if the factor $T[i..j]$ corresponding to DE is an Ab-square centred in $k = |C|$. The figure shows the case when k is in the right half of the strings; the other case is symmetric.

By Fact 1 the cost of the algorithm can be given by a recurrence

$$S(n) = 2 \cdot S\left(\frac{n}{2}\right) + \mathcal{O}(n^{2-\epsilon})$$

which results in $S(n) = \mathcal{O}(n^{2-\epsilon})$ for $\epsilon > 0$.

In case of odd Ab-squares let

$$P_{l,r}^c(W) = \{\text{Parikh}(W[1..k]) : l \leq k \leq r, k \bmod 2 = c\}.$$

In the algorithm the statement $\mathcal{M} = \{|C| : 2C \in (\mathcal{A} + \mathcal{B}) \cap 2 \cdot \mathcal{C}\}$ is executed for both $c \in \{0, 1\}$, with

$$\mathcal{A} = P_{0,m-1}^c(T), \quad \mathcal{B} = P_{m,n}^c(T), \quad \mathcal{C} = P_{0,n}^{1-c}(T).$$

Other parts of the algorithm, as well as its analysis, are essentially the same. \blacktriangleleft

6 Detecting odd Ab-squares

Unfortunately the string T from Lemma 19 has many Ab-squares which are not well-placed. Our approach is to embed the (slightly) modified string T into a string which is a special composition of T and a combination of long quaternary Ab-square-free strings. The resulting string will fix the potential centers in specified locations. We use additional letters: \diamond , \circ and $0, \dots, 6$.

We show first a fact useful in fixing Ab-squares in specified places (Lemma 26). Let P_{t-2} be any Ab-square-free string of length $t-2$ over alphabet $\{3, 4, 5, 6\}$ (it can be constructed using Keränen's construction [27] of quaternary Ab-square-free strings). Let us define

$$U_{2t} = 0 P_{t-2} 1 2 P_{t-2}^R 0.$$

The following lemma is proved in the full version of the paper.

► **Lemma 24.** *The string $(U_{2t})^m$ contains exactly the following Ab-squares:*

- (1) *of length divisible by $4t$; and*
- (2) *with the center between two 0's and of all admissible even lengths other than $(4q+2)t$, for an integer $q \geq 0$.*

For equal-length strings X, Y we define the string

$$\text{shuffle}_{\diamond}(X, Y) = X[1] \diamond Y[1] X[2] \diamond Y[2] X[3] \diamond Y[3] \dots$$

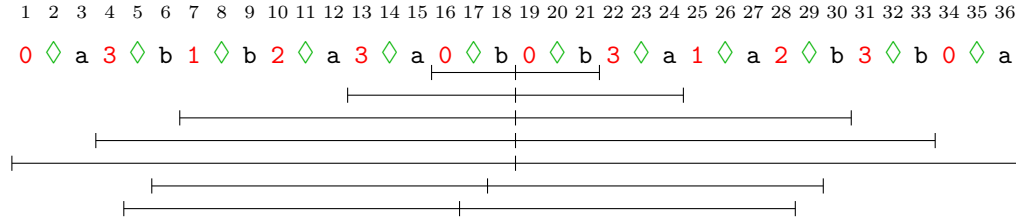
For example, $\text{shuffle}_{\diamond}(\text{abc}, \text{ABC}) = \text{a} \diamond \text{Ab} \diamond \text{Bc} \diamond \text{C}$.

The parity condition for half lengths of Ab-squares in the following observation justifies the usage of the additional letter \diamond in shuffle . Let $U_{[X]}$ be the string resulting from U by removing all letters outside $\text{Alph}(X)$.

► **Observation 25.** *Assume X, Y are equal-length strings composed of disjoint sets of letters distinct from \diamond and W is an Ab-square in $\text{shuffle}_{\diamond}(X, Y)$. Then $W_{[X]}, W_{[Y]}$ are Ab-squares in X, Y , respectively (we say that these Ab-squares are implied by W). Moreover, $|W_{[X]}|/2, |W_{[Y]}|/2, |W|/2$ are of the same parity.*

We say that an even-length factor of a string X is *centred at i* if it has its center between positions i and $i+1$ in X . By $a \mid b$ and $a \nmid b$ we denote that a divides b and a does not divide b . For an illustration of the following lemma, see Figure 6.

► **Lemma 26.** *Let $X = (U_{2t})^{n-1}$, Y be a string of length $|X|$ such that its alphabet is disjoint with $\text{Alph}(X) \cup \{\diamond\}$, $W = \text{shuffle}_{\diamond}(X, Y)$, and let an integer ℓ satisfy $12t \nmid \ell$. Then a length- ℓ factor of W is an Ab-square if and only if it is centred in W at $r \equiv \{0, -1, -2\} \pmod{6t}$, Y contains an Ab-square factor of length $\ell/3$ centred in Y at $\lfloor r/3 \rfloor$, and $6t \nmid \ell$.*



■ **Figure 6** Illustration of Lemma 26. Let $X = (U_6)^2$. The string $Y = (\text{abba})^3$, composed of black letters, contains many Ab-squares. However the string $Z = \text{shuffle}_\diamond(X, Y)$ of length 36, shown above, contains only Ab-squares centred at 16, 17 or 18, as in the figure. The implied Ab-squares in Z are only those which are centred at positions 5 or 6 in Z .

Proof. By the disjointness of sets of letters in X, Y and $\{\diamond\}$, each Ab-square in W has length that is divisible by 3. The following claim is then readily verified (cf. Observation 25).

▷ **Claim 27.** For positive integer ℓ such that $6 \mid \ell$, a length- ℓ factor of W centred at r is an Ab-square if and only if the length- $\ell/3$ factors in X and Y centred at $\lfloor \frac{r+2}{3} \rfloor$ and $\lfloor \frac{r}{3} \rfloor$, respectively, are Ab-squares.

Let integer $\ell > 0$ satisfy $6 \mid \ell$ and $12t \nmid \ell$. We show two implications.

(\Rightarrow) If W contains an Ab-square factor of length ℓ centred at some r , then the implied Ab-square factor of X has length $\ell/3$, where $4t \nmid \ell/3$, so by Lemma 24 it has its center between two 0's, i.e., $2t \mid \lfloor \frac{r+2}{3} \rfloor$. Hence, $r \equiv \{0, -1, -2\} \pmod{6t}$.

Moreover, $2t \nmid \ell/3$ also by Lemma 24. Finally, the implied Ab-square factor of Y indeed has length $\ell/3$ and is centred at $\lfloor r/3 \rfloor$.

(\Leftarrow) Let $r \equiv \{0, -1, -2\} \pmod{6t}$, $6t \nmid \ell$, and assume that Y contains an Ab-square factor of length $\ell/3$ centred at $\lfloor r/3 \rfloor$. We have $2t \mid \lfloor \frac{r+2}{3} \rfloor$ and $2t \nmid \ell/3$, so by Lemma 24 the string X contains an Ab-square factor of length $\ell/3$ centred at $\lfloor \frac{r+2}{3} \rfloor$. Finally, the unary string \diamond^{2tm} , certainly contains an Ab-square factor of length $\ell/3$ centred at $\lfloor \frac{r+1}{3} \rfloor$. By the claim, W contains an Ab-square of length ℓ centred at r that implies the three Ab-squares. ◀

► **Theorem 28.** *Checking if a length- n string over an alphabet of size $\omega(1)$ contains an odd Ab-square is 3SUM-hard. Moreover, for a string over an alphabet of size $14 + k$, for a constant k , the same problem cannot be solved in $\mathcal{O}(n^{2 - \frac{6}{3+k} - \varepsilon})$ time, for a constant $\varepsilon > 0$, unless the 3SUM conjecture fails.*

Proof. We use the technique of fixing Ab-squares from Lemma 26. Moreover, we make the following minor modifications upon the construction of Section 4:

- (1) Each fragment \mathbf{b}^{kd} is extended by one letter to \mathbf{b}^{kd+1} , and
- (2) the letters \bullet, \star are replaced each by two letters $\bullet\circ, \star\circ$, respectively.

Intuitively, (1) allows to extend Ab-squares considered in the proof of Lemma 19 by one letter \mathbf{b} to either side, and (2) makes $|\mathbf{A}| = |\mathbf{B}| = |\mathbf{S}_i^I|$ even which facilitates the usage of Lemma 26 with $Y = T$. It can be verified by inspecting the proof that Lemma 19 still holds after these two changes. We refer to all the notions from Section 4 after these modifications.

It is enough now to show the following claim for $X = (U_{2t})^{n-1}$, where $2t = |T|/(n-1)$. We assume that $n \geq 3$.

▷ **Claim 29.** An odd-half instance of 3DAP is a YES-instance if and only if $W = \text{shuffle}_\diamond(X, T)$ has an odd Ab-square factor.



■ **Figure 7** A schematic structure of a fragment of T after insertion of symbols \circ . There are $3(n-1)$ (underbraced) blocks in T , each of size $M+2$, and $2t = 3M+6$.

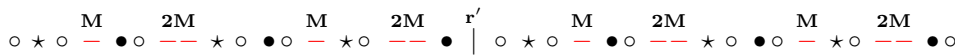
Proof. (\Rightarrow) Assume that \bar{x} is an odd-half instance and $3\text{DAP}(\bar{x})$ has a solution.

By Lemma 19, T contains a well-placed Ab-square, that is, an Ab-square centred at a position r' such that $2t \mid r'$. (Recall that $3(M+2) = 2t$.) Moreover, in the proof of that lemma it is shown that in this case there exists a well-placed Ab-square in T that satisfies the following additional requirements: **(1)** it starts within the gadget **B**; **(2)** it starts and ends within a block of **b**'s; **(3)** its maximal prefix and suffix consisting of letters **b** are \mathbf{b}^e and \mathbf{b}^f , where $e, f \leq kd$.

Let ℓ' denote the half length of this Ab-square. By **(2)** and **(3)**, if ℓ' is even, the Ab-square can be extended by one letter \mathbf{b} to either side (because we have extended each block \mathbf{b}^{kd}) so that ℓ' becomes odd. Moreover, by **(1)**, we have $\ell' \bmod (2t) \in [\frac{4}{3}t, 2t)$, in particular, $t \nmid \ell'$. Then Lemma 26 concludes that the factor of W centred at $r = 3r' \equiv 0 \pmod{6t}$ and of length $6\ell'$ such that $6t \nmid 6\ell'$ is an Ab-square. Its half length, $3\ell'$, is odd, as desired.

(\Leftarrow) Assume that W has an Ab-square factor U of length ℓ such that $\ell/2$ is odd. In particular, we have $12t \nmid \ell$, so by Lemma 26 the Ab-square U is centred in W at $r \equiv \{0, -1, -2\} \pmod{6t}$ and T contains an Ab-square factor V of length $\ell/3$ centred in T at $r' = \lfloor r/3 \rfloor$. If $6t \mid r$, then $2t \mid r'$ and V is well-placed.

Otherwise, V cannot be an Ab-square due to the following fact: T does not contain an Ab-square factor of length ℓ not divisible by $4t$ and centred at $r' \equiv -1 \pmod{2t}$. Indeed, similarly as in the proof of Lemma 19, we will show that each even-length factor centred at such r' contains different counts of one of the letters \bullet, \star, \circ in both halves. The positions of letters \bullet, \star, \circ in T repeat with period $6(M+2)$, so it is sufficient to inspect the first 6 blocks on each side, as the remaining ones will behave periodically; see Figures 7 and 8. An exhaustive verification of several cases can be performed by counting distances of letters from $\{\bullet, \star, \circ\}$ in both directions to the center of the factor (for more details see the full version).



■ **Figure 8** A simplified version of Figure 7. Which position in a block can be the starting position of an Ab-square with the center at r' (one position to the left of a *good* center), only counting ●, ★, ○.

Consequently, as in Lemma 19, the corresponding instance of 3DAP is a YES-instance.

△

The complexities in the theorem are obtained as in Theorems 20 and 21.

7 Open problems

The most interesting questions that remain open are as follows:

1. Is checking Ab-square-freeness 3SUM-hard? Our reductions allowed us to show 3SUM-hardness of detecting an odd Ab-square.

2. Can one detect an additive square in a length- n string over a constant-sized alphabet in $\mathcal{O}(n^{2-\varepsilon})$ time, for some $\varepsilon > 0$? We have shown 3SUM-hardness of this problem for an alphabet that is polynomial in n .


References

1. Peyman Afshani, Ingo van Duijn, Rasmus Killmann, and Jesper Sindahl Nielsen. A lower bound for jumbled indexing. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 592–606. SIAM, 2020. doi:10.1137/1.9781611975994.36.
2. Amihood Amir, Alberto Apostolico, Tirza Hirst, Gad M. Landau, Noa Lewenstein, and Liat Rozenberg. Algorithms for jumbled indexing, jumbled border and jumbled square on run-length encoded strings. *Theoretical Computer Science*, 656:146–159, 2016. doi:10.1016/j.tcs.2016.04.030.
3. Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. 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 114–125. Springer, 2014. doi:10.1007/978-3-662-43948-7_10.
4. Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.
5. Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3.
6. Felix Adalbert Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America*, 32(12):331–332, 1946. doi:10.1073/pnas.32.12.331.
7. Tom C. Brown and Allen R. Freedman. Arithmetic progressions in lacunary sets. *Rocky Mountain Journal of Mathematics*, 17(3):587–596, 1987.
8. Tom C. Brown, Veselin Jungić, and Andrew Poelstra. On double 3-term arithmetic progressions. *Integers*, 14:A43, 2014. URL: <https://www.emis.de/journals/INTEGERS/papers/o43/o43.Abstract.html>.
9. Julien Cassaigne, James D. Currie, Luke Schaeffer, and Jeffrey O. Shallit. Avoiding three consecutive blocks of the same size and same sum. *Journal of the ACM*, 61(2):10:1–10:17, 2014. doi:10.1145/2590775.
10. Timothy M. Chan and Qizheng He. Reducing 3SUM to Convolution-3SUM. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.
11. 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.
12. Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.
13. Larry J. Cummings and William F. Smyth. Weak repetitions in strings. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 24:33–48, 1997.

- 14 Bartłomiej Dudek, Paweł Gawrychowski, and Tatiana Starikovskaya. All non-trivial variants of 3-LDT are equivalent. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 974–981. ACM, 2020. doi:10.1145/3357713.3384275.
- 15 Roger C. Entringer, Douglas E. Jackson, and J.A. Schatz. On nonrepetitive sequences. *Journal of Combinatorial Theory, Series A*, 16(2):159–164, 1974. doi:10.1016/0097-3165(74)90041-7.
- 16 Paul Erdős. Some unsolved problems. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 6:221–254, 1961.
- 17 Jeff Erickson. Finding longest arithmetic progressions, 1999. URL: <https://jeffe.cs.illinois.edu/pubs/arith.html>.
- 18 Aleksandr Andreevich Evdokimov. Strongly asymmetric sequences generated by a finite number of symbols. *Doklady Akademii Nauk SSSR*, 179(6):1268–1271, 1968.
- 19 Gabriele Fici, Filippo Mignosi, and Jeffrey O. Shallit. Abelian-square-rich words. *Theoretical Computer Science*, 684:29–42, 2017. doi:10.1016/j.tcs.2017.02.012.
- 20 Gabriele Fici and Aleks Saarela. On the minimum number of abelian squares in a word. In Maxime Crochemore, James Currie, Gregory Kucherov, and Dirk Nowotka, editors, *Combinatorics and Algorithmics of Strings (Dagstuhl Seminar 14111)*, volume 4 (3), pages 34–35, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/DagRep.4.3.28.
- 21 Aviezri S. Fraenkel, Jamie Simpson, and Mike Paterson. On weak circular squares in binary words. In Alberto Apostolico and Jotun Hein, editors, *Combinatorial Pattern Matching, 8th Annual Symposium, CPM 97, Aarhus, Denmark, June 30 - July 2, 1997, Proceedings*, volume 1264 of *Lecture Notes in Computer Science*, pages 76–82. Springer, 1997. doi:10.1007/3-540-63220-4_51.
- 22 Allen R. Freedman and Tom C. Brown. Sequences on sets of four numbers. *Integers*, 16:A33, 2016. URL: <http://math.colgate.edu/~integers/q33/q33.Abstract.html>.
- 23 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How hard is it to find (honest) witnesses? In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPICs*, pages 45:1–45:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.45.
- 24 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.
- 25 Lorenz Halbeisen and Norbert Hungerbühler. An application of van der Waerden’s theorem in additive number theory. *Integers*, 0:A7, 2000. URL: <http://math.colgate.edu/~integers/a7/a7.pdf>.
- 26 Veikko Keränen. A powerful abelian square-free substitution over 4 letters. *Theoretical Computer Science*, 410(38-40):3893–3900, 2009. doi:10.1016/j.tcs.2009.05.027.
- 27 Veikko Keränen. Abelian squares are avoidable on 4 letters. In Werner Kuich, editor, *Automata, Languages and Programming, ICALP 1992*, volume 623 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 1992. doi:10.1007/3-540-55719-9_62.
- 28 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Maximum number of distinct and nonequivalent nonstandard squares in a word. *Theoretical Computer Science*, 648:84–95, 2016. doi:10.1016/j.tcs.2016.08.010.
- 29 Tomasz Kociumaka, Jakub Radoszewski, and Bartłomiej Wiśniewski. Subquadratic-time algorithms for abelian stringology problems. In Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, volume 9582 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2015. doi:10.1007/978-3-319-32859-1_27.

- 30 Tomasz Kociumaka, Jakub Radoszewski, and Bartłomiej Wiśniewski. Subquadratic-time algorithms for abelian stringology problems. *AIMS Medical Science*, 4(3):332–351, 2017. doi:10.3934/ms.2017.3.332.
- 31 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.ch89.
- 32 Florian Lietard and Matthieu Rosenfeld. Avoidability of additive cubes over alphabets of four numbers. In Natasa Jonoska and Dmytro Savchuk, editors, *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings*, volume 12086 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2020. doi:10.1007/978-3-030-48516-0_15.
- 33 Giuseppe Pirillo and Stefano Varricchio. On uniformly repetitive semigroups. *Semigroup Forum*, 49:125–129, 1994. doi:10.1007/BF02573477.
- 34 Peter A. B. Pleasants. Non-repetitive sequences. *Mathematical Proceedings of the Cambridge Philosophical Society*, 68:267–274, 1970.
- 35 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. 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 603–610. ACM, 2010. doi:10.1145/1806689.1806772.
- 36 Michaël Rao and Matthieu Rosenfeld. Avoiding two consecutive blocks of same size and same sum over \mathbb{Z}^2 . *SIAM Journal on Discrete Mathematics*, 32(4):2381–2397, 2018. doi:10.1137/17M1149377.
- 37 Lawrence Bruce Richmond and Jeffrey O. Shallit. Counting abelian squares. *Electronic Journal of Combinatorics*, 16(1), 2009. URL: http://www.combinatorics.org/Volume_16/Abstracts/v16i1r72.html.
- 38 Raphaël Salem and Donald C. Spencer. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America*, 28(12):561–563, 1942. doi:10.1073/pnas.28.12.561.
- 39 Jamie Simpson. Solved and unsolved problems about abelian squares, 2018. arXiv:1802.04481.
- 40 Shiho Sugimoto, Naoki Noda, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing abelian string regularities based on RLE. In Ljiljana Brankovic, Joe Ryan, and William F. Smyth, editors, *Combinatorial Algorithms - 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, July 17-21, 2017, Revised Selected Papers*, volume 10765 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2017. doi:10.1007/978-3-319-78825-8_34.

On Approximate Compressions for Connected Minor-Hitting Sets

M. S. Ramanujan   

University of Warwick, Coventry, UK

Abstract

In the CONNECTED \mathcal{F} -DELETION problem, \mathcal{F} is a fixed finite family of graphs and the objective is to compute a minimum set of vertices (or a vertex set of size at most k for some given k) such that (a) this set induces a connected subgraph of the given graph and (b) deleting this set results in a graph which excludes every $F \in \mathcal{F}$ as a minor. In the area of kernelization, this problem is well known to exclude a polynomial kernel subject to standard complexity hypotheses even in very special cases such as $\mathcal{F} = \{K_2\}$, i.e., CONNECTED VERTEX COVER.

In this work, we give a $(2 + \epsilon)$ -approximate polynomial compression for the CONNECTED \mathcal{F} -DELETION problem when \mathcal{F} contains at least one planar graph. This is the first approximate polynomial compression result for this generic problem. As a corollary, we obtain the first approximate polynomial compression result for the special case of CONNECTED η -TREEWIDTH DELETION.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Mathematics of computing \rightarrow Approximation algorithms

Keywords and phrases Parameterized Complexity, Kernelization, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.78

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 this area is that of a *kernelization* (also called a *kernel*), 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. Over the last decade, the area of kernelization has seen the development of a wide range of tools to design preprocessing algorithms and lower bounds techniques. The reader may find an introduction to the field in [24, 25, 3, 7].

An “efficient preprocessing algorithm” in this setting is referred to as a *polynomial kernel* and is simply a kernel whose output has size bounded polynomially in a parameter of the input. The central classification task in the area of kernelization is to identify NP-hard problems and associated parameters for which polynomial kernels exist and one of the main success stories in the area is the development of a rich theory of lower bounds based on complexity theoretic assumptions [1, 5, 2, 18, 4, 9, 21, 6, 20] allowing one to rule out the existence of polynomial kernels completely or lower bound the degree of the polynomial.

One fundamental class of problems for which polynomial kernels have been ruled out in this way is the class of subgraph hitting problems *with connectivity constraints*. It is well-known that placing connectivity constraints on subgraph hitting problems can have a dramatic effect on their amenability to efficient preprocessing. A case in point is the classic VERTEX COVER problem. This problem is known to admit a kernel whose output has $\mathcal{O}(k)$ vertices [3], where the parameter k is the solution size. However, the CONNECTED VERTEX COVER problem is amongst the earliest problems shown to exclude a polynomial kernel [6] and this lower bound immediately rules out the possibility of such a kernel for numerous well-studied generalizations of this problem. Consequently, one cannot hope to design approximation



© M. S. Ramanujan;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 78; pp. 78:1–78:16

Leibniz International Proceedings in Informatics



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

algorithms for such problems via polynomial kernels in the traditional sense and so, 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 in general and in the design of approximation algorithms for connectivity constrained subgraph hitting problems.

One of the most frequently investigated subgraph hitting problems in the literature is the \mathcal{F} -DELETION problem which generalizes numerous well-studied NP-complete problems. In this problem, \mathcal{F} is a fixed finite family of graphs and one is given a graph G and an integer k as input. The objective is to determine whether at most k vertices can be deleted from G so that the resulting graph is \mathcal{F} -minor free (does not contain a minor isomorphic to a graph in \mathcal{F}). The optimization version of this problem asks for a minimum set of vertices whose deletion leaves a graph which is \mathcal{F} -minor free. Well-studied special cases of this problem include VERTEX COVER ($\mathcal{F} = \{K_2\}$), FEEDBACK VERTEX SET ($\mathcal{F} = \{K_3\}$), PLANARIZATION ($\mathcal{F} = \{K_{3,3}, K_5\}$) [27], DIAMOND HITTING SET ($\mathcal{F} = \{\theta_3\}$) [14], PATHWIDTH ONE VERTEX DELETION ($\mathcal{F} = \{K_3, T_2\}$) [29], and θ_c -DELETION [22, 15]. A common feature shared by many such well explored special cases of this problem is that \mathcal{F} contains at least one planar graph. Motivated by this, Fomin et al. [17] investigated this restricted variant of the problem (when the family \mathcal{F} contains at least one planar graph) and obtained a polynomial kernel for every such \mathcal{F} . This particular variant of \mathcal{F} -DELETION is known in the literature as the PLANAR \mathcal{F} -DELETION problem.

Motivated by the prevalence of the PLANAR \mathcal{F} -DELETION problem in existing work on subgraph hitting problems, we initiate the study of the PLANAR \mathcal{F} -DELETION problem when there is a connectivity constraint on the solution. This problem, which we call the CONNECTED PLANAR \mathcal{F} -DELETION problem, is formally defined as follows. The input is a graph G , and integer k (the parameter) and the goal is to determine whether there is a set $S \subseteq V(G)$ of size at most k such that $G[S]$ is connected and $G - S$ is \mathcal{F} -minor free? The set S is called a connected \mathcal{F} -deletion set.

As already discussed, CONNECTED PLANAR \mathcal{F} -DELETION displays stark differences to the version without connectivity constraints when considering the approximability as well as amenability to efficient preprocessing even when \mathcal{F} is a very simple family such as $\{K_2\}$. To be specific, since this problem is a clear generalization of CONNECTED VERTEX COVER, it cannot have a $(2 - \varepsilon)$ -approximation in polynomial time for any fixed $\varepsilon > 0$ under UGC [23] and moreover, it is unlikely to have a polynomial kernel [6].

Since using the existing notion of polynomial kernels and associated reduction rules in order to design approximation algorithms for such connectivity constrained problems appears to be difficult, we rely on the recently developed notion of α -approximate kernels which was introduced by Lokshtanov et al. [26] in order to facilitate the rigorous analysis of preprocessing algorithms in conjunction with approximation algorithms.

Informally speaking, an α -approximate kernel is a polynomial-time algorithm that given as input a pair (I, k) where I is the problem instance and k is the parameter, outputs an instance (I', k') of the same problem such that $|I'| + k' \leq g(k)$ for some computable function g and any c -approximate solution for the instance I' can be turned in polynomial time into a $(c \cdot \alpha)$ -approximate solution for the original instance I . When the output is an instance of a different problem (with the remaining conditions holding), one obtains an α -approximate compression.

As earlier, the notion of efficiency in this context is captured by the function g being required to be 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 kernel 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 CONNECTED VERTEX COVER admits an α -approximate polynomial kernel for every $\alpha > 1$. Their result provided a promising starting point towards a refined understanding of the role played by connectivity constraints in relation to preprocessing for covering problems on graphs. Subsequently, Eiben et al. [11] extended this result to the CONNECTED \mathcal{H} -HITTING SET problem where \mathcal{H} is a fixed collection of finite subgraphs and the solution is a minimum set of vertices in the given graph G which induce a connected subgraph and hit all copies of graphs in \mathcal{H} which are present in G . Recently, Ramanujan [30] obtained the first α -approximate polynomial kernel (for every $\alpha > 1$) for the CONNECTED FEEDBACK VERTEX SET problem, demonstrating the power of approximate preprocessing for cases where the goal is to hit obstructions of unbounded size while maintaining connectivity of the hitting set.

Our results

A formal definition of α -approximate kernels and compressions can be found in Section 2.

► **Theorem 1.** *For every fixed $0 < \varepsilon < 1$, CONNECTED PLANAR \mathcal{F} -DELETION has a $(2 + \varepsilon)$ -approximate compression of polynomial size.*

As an immediate corollary of Theorem 1, we obtain a factor- $(2 + \varepsilon)$ parameterized approximation algorithm for CONNECTED PLANAR \mathcal{F} -DELETION running in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ for every fixed $0 < \varepsilon < 1$. In fact, the proof techniques we use in order to prove Theorem 1 also enable us to obtain a polynomial-time $\text{poly}(\text{OPT})$ approximation for this problem.

► **Theorem 2.** *For every fixed \mathcal{F} containing a planar graph, 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 \mathcal{F} -deletion set of size at most k or returns a connected \mathcal{F} -deletion set of G of size $k^{\mathcal{O}(1)}$.*

Using Theorem 2 and adopting an approach similar to that in [30], we obtain the following.

► **Theorem 3.** *There is a $0 < \delta < 1$ such that CONNECTED PLANAR \mathcal{F} -DELETION can be approximated within a factor $\min\{\text{OPT}^{\mathcal{O}(1)}, n^{1-\delta}\}$ in polynomial time.*

Related work on approximation for connected hitting set problems

Grigoriev and Sitters [19] 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. Fiorini et al. [14] studied the DIAMOND HITTING SET problem (where $\mathcal{F} = \{\theta_3\}$) and obtained the first constant-factor approximation.

¹ We note that this problem can be easily seen to be fixed-parameter tractable parameterized by k since the problem is MSO-expressible and the treewidth of yes-instances must be bounded by $\mathcal{O}(k)$, allowing for an invocation of Courcelle's theorem.

2 Preliminaries

For a graph G , we denote by $\mathcal{CC}(G)$ the set of connected components of G . 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 . For a set $X \subseteq V(G)$, the graph obtained from G by identifying the vertices in X is the graph G' defined as follows. The vertex set of G' is $(V(G) \setminus X) \cup \{x\}$ where $x \notin V(G)$. For every edge of G which is not incident on X , G' has the same edge. For every edge $(u, v) \in E(G)$ where $u \in X, v \notin X$, we add a new edge (x, v) . Note that we ignore all edges which have both endpoints in X .

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 (x, k) , where k is called the parameter. We assume that k is given in unary and hence $k \leq |x|$. The notion of kernelization is formally defined as follows.

► **Definition 4 (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 5 ([26]).** 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 the problems we come across in this paper are minimization problems, we state some of the definitions only in terms of minimization problems when the definition for maximization problems is analogous. For instance, the parameterized optimization version of CONNECTED PLANAR \mathcal{F} -DELETION is defined as follows (using the convention from [26]). This is a minimization problem with the optimization function $\text{CPFD} : \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\infty\}$ defined as follows.

$$\text{CPFD}(G, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a feasible solution,} \\ \min\{|S|, k + 1\} & \text{otherwise.} \end{cases}$$

► **Definition 6 ([26]).** For a parameterized minimization problem Π , the optimum value of an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ is $\text{OPT}_\Pi(I, k) = \min_{\substack{s \in \Sigma^* \\ |s| \leq |I| + k}} \Pi(I, k, s)$.

We now recall the other relevant definitions from [26] regarding approximate kernels.

► **Definition 7 ([26]).** 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 $\frac{\Pi(I, k, s)}{\text{OPT}_{\Pi}(I, k)} \leq \alpha \cdot \frac{\Pi(I', k', s')}{\text{OPT}_{\Pi'}(I', k')}$.

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 8** ([26]). Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized minimization problems. An α -approximate polynomial parameter transformation (α -appt for short) \mathcal{A} from Π to Π' is a pair of polynomial-time algorithms, called reduction algorithm $\mathcal{R}_{\mathcal{A}}$ and solution-lifting algorithm. Given as input an instance (I, k) of Π the reduction algorithm outputs an instance (I', k') of Π' such that $k' = k^{O(1)}$. The solution-lifting algorithm takes as input an instance (I, k) of Π , the output instance $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$ of Π' , and a solution s' to the instance I' and outputs a solution s to (I, k) such that

$$\frac{\Pi(I, k, s)}{\text{OPT}_{\Pi}(I, k)} \leq \alpha \cdot \frac{\Pi'(I', k', s')}{\text{OPT}_{\Pi'}(I', k')}.$$

► **Definition 9** ([26], α -approximate compression). Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized minimization problems. An α -approximate compression from Π to Π' is an α -appt \mathcal{A} from Π to Π' such that $\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{R}_{\mathcal{A}}(I, k), I \in \Sigma^*\}$, is upper bounded by a computable function $g : \mathbb{N} \rightarrow \mathbb{N}$, where $\mathcal{R}_{\mathcal{A}}$ is the reduction algorithm in \mathcal{A} . We say that \mathcal{A} is an α -approximate polynomial compression if g is a polynomial function.

Treewidth, t -Boundaried graphs and minors

We now recall standard definitions regarding treewidth and minor models. The notation is based on [17]. Let G be a graph. A *tree decomposition* of G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ where T is a tree and \mathcal{X} is a collection of subsets of $V(G)$ such that (a) $\forall e = uv \in E(G), \exists t \in V(T) : \{u, v\} \subseteq X_t$ and (b) $\forall v \in V(G), T[\{t \mid v \in X_t\}]$ is a non-empty connected subtree of T . We call the vertices of T *nodes* and the sets in \mathcal{X} *bags* of the tree decomposition (T, \mathcal{X}) . The *width* of (T, \mathcal{X}) is denoted by $\text{width}(T, \mathcal{X})$ is defined as $\max\{|X_t| - 1 \mid t \in V(T)\}$ and the *treewidth* of G is the minimum width over all tree decompositions of G . A t -boundaried graph is a graph G and a set $B \subset V(G)$ of size at most t with each vertex $v \in B$ having a label $\ell_G(v) \in \{1, \dots, t\}$. Each vertex in B has a unique label. We refer to B as the *boundary* of G . We use the notation (G, B) to refer to the t -boundaried graph G with boundary B .

Least Common Ancestor-Closure of Sets in Graphs of Bounded Treewidth

For a graph G with a nice tree decomposition (T, χ) rooted at $r \in V(T)$ and vertex set $X \subseteq V(G)$ the least common ancestor-closure $\text{LCA-closure}(T, \chi, X)$ is defined as follows. We let $M \subseteq V(T)$ denote a minimal set of nodes in T such that for every $x \in X$, there is a bag $u \in M$ such that $x \in \chi(u)$ and moreover, u is the closest such vertex to r . Finally, we define $\text{LCA-closure}(T, \chi, X)$ as the set $\chi(\text{LCA-closure}(M))$, where $\text{LCA-closure}(M)$ denotes the LCA-closure of M in the rooted tree T . We ignore the explicit reference to the root r in the notation $\text{LCA-closure}(T, \chi, X)$ because we will be using an arbitrary vertex as the root when invoking this definition.

The following lemma is a direct consequence of the definition of $\text{LCA-closure}(T, \chi, X)$ and the application of LCA-closure on trees (see, for example, [17]).

► **Lemma 10.** *Let G be a graph with a nice tree decomposition (T, χ) and let $X \subseteq V(G)$. Let X' denote the set $\text{LCA-closure}(T, \chi, X)$. Then, $|X'| \leq 2|X| \cdot \text{width}(T, \chi)$ and for every connected component C of $G - X'$, $|N(C)| \leq 2 \cdot \text{width}(T, \chi)$.*

► **Definition 11.** *Let G be a graph and H be a minor of G with $V(H) = \{h_1, \dots, h_\ell\}$ and suppose that H has no self-loops. A set $\mathcal{P}_H = \{P_1, \dots, P_\ell\}$ of pairwise vertex-disjoint subsets of $V(G)$ is said to be a minor model of H in G if*

- $G[P_i]$ is connected for every $i \in [\ell]$ and
- there is an injective mapping $\phi : E(H) \rightarrow E(G)$ such that for every $e = (i, j) \in E(H)$, $\phi(e)$ is an edge in G with one endpoint in P_i and one in P_j .

Note that if H has parallel edges, then the minor model needs to have a unique edge corresponding to each parallel edge. We say that H is a minimal minor model if there is no strict subgraph of $G[\bigcup_{i \in [\ell]} P_i]$ which also contains H as a minor.

► **Definition 12.** *Let G_1 and G_2 be two graphs, and let t be a fixed positive integer. For $i \in \{1, 2\}$, let f_{G_i} be a function that associates with every vertex of $V(G_i)$ some subset of $[t]$. The image of a vertex $v \in G_i$ under f_{G_i} is called the label of that vertex. We say that G_1 is label-wise isomorphic to G_2 , and denote it by $G_1 \cong_t G_2$, if there is a map $h : V(G_1) \rightarrow V(G_2)$ such that (a) h is one to one and onto; (b) $(u, v) \in E(G_1)$ if and only if $(h(u), h(v)) \in E(G_2)$ and (c) $f_{G_1}(v) = f_{G_2}(h(v))$. We call h a label-preserving isomorphism.*

Notice that the first two conditions of Definition 12 simply indicate that G_1 and G_2 are isomorphic. Now, let G be a t -boundaried graph, that is, G has t distinguished vertices, uniquely labeled from 1 to t . Given a t -boundaried graph G , we define a canonical labeling function $\mu_G : V(G) \rightarrow 2^{[t]}$. The function μ_G maps every distinguished vertex v with label $\ell \in [t]$ to the set $\{\ell\}$, that is $\mu_G(v) = \{\ell\}$, and for all other vertices we have that $\mu_G(v) = \emptyset$.

Next we define a notion of labeled edge contraction. Let H be a graph together with a function $f_H : V(H) \rightarrow 2^{[t]}$ and $(u, v) \in E(H)$. Furthermore, let H' be the graph obtained from H by identifying the vertices u and v into w_{uv} and removing all loops. Then by *labeled edge contraction* of an edge (u, v) of a graph H , we mean obtaining a graph H' with the label function $f_{H'} : V(H') \rightarrow 2^{[t]}$ defined as follows. For $x \in V(H') \cap V(H)$ we have that $f_{H'}(x) = f_H(x)$ and for w_{uv} we define $f_{H'}(w_{uv}) = f_H(u) \cup f_H(v)$. Now we introduce a notion of labeled minors of a t -boundaried graph.

► **Definition 13.** *Let H be a graph together with a function $f : V(H) \rightarrow 2^{[t]}$ and (G, Z) be a t -boundaried graph with canonical labeling function μ_G . A graph H is called a labeled minor of G , if we can obtain a labeled isomorphic copy of H from G by performing edge deletions and labeled edge contractions. The h -folio of a t -boundaried graph (G, Z) is the set $\mathcal{M}_h(G, Z)$ of all labeled minors of G (starting with the canonical labeling on G) on at most h vertices.*

We also need the following well-known result bounding the treewidth of any graph which excludes a fixed planar graph as a minor.

► **Proposition 14 ([17]).** *For every fixed planar graph H , there is a constant λ_H such that any graph G with $\text{tw}(G) \geq \lambda_H$ contains H as a minor.*

Steiner Trees

For a graph G , a set $R \subseteq V(G)$ called *terminals* and a cost function $w : E(G) \rightarrow \mathbb{N} \cup \{0\}$, a *Steiner tree* is a subtree T of G such that $R \subseteq V(T)$, and the *cost* of a tree T is defined as $w(T) = \sum_{e \in E(T)} w(e)$. In the STEINER TREE problem we are given as input the graph G ,

the set R and the cost function $w : E(G) \rightarrow \mathbb{N} \cup \{0\}$. The task is to find an *optimal Steiner tree*, which is a Steiner tree of minimum cost. However, in this paper we will only work with edges of unit or zero cost and we denote by w_1 the function that assigns a cost of 1 to every edge of the graph under consideration. When T is a Steiner tree for the set of terminals R , we say that T is an *R -Steiner tree*. An R -Steiner tree T is said to be *minimal* if there is no $e \in E(T)$ such that $T - e$ also contains an R -Steiner tree. It is well-known that there is an algorithm for STEINER TREE with a single exponential dependence on the number of terminals.

► **Proposition 15** ([8, 28]). *The STEINER TREE problem can be solved in time $2^{O(|R|)} n^{O(1)}$.*

Let R continue to denote the set of terminals. For a $k \in \mathbb{N}$, a *k -component* is a tree with at most k leaves, all of which are in R . A *k -restricted Steiner tree* \mathcal{T} is a collection of k -components, such that the union of these components is a Steiner tree T . The cost of \mathcal{T} is the sum of the costs of all the k -components in \mathcal{T} . It may be the case that multiple k -components which are part of a k -restricted Steiner tree, share edges. As a result, some edges may contribute multiple times to the cost of \mathcal{T} . To keep the presentation simple, if the k -components of a k -restricted Steiner tree are clear from the context, then we also refer to the Steiner tree composed of their union as a k -restricted Steiner tree. Recall that according to its original definition, a k -restricted Steiner tree is a *set* of k -components.

3 Overview of our Algorithms

Fix $0 < \varepsilon < 1$. We identify a partition (A, B, C) of the vertex set of G , where $|B| = k^{O(f(1/\varepsilon))}$ and there are no edges between A and C . In other words, B separates A and C . We then prove that the vertices in C only play the role of “connectors” and removing them from a hypothetical solution S may disconnect $G[S]$, but will still leave a minimal hitting set for all \mathcal{F} -minor models. On the other hand, while the interaction of vertices in A with the solution S could be much more complex, we show that the number of connected components of $G[A]$ is $k^{O(f(1/\varepsilon))}$ and these can be shown to have a well-structured neighborhood in B . Once we have this partition in hand, we focus on each connected component of $G[A]$ separately and from each component we identify a set of $k^{O(f(1/\varepsilon))}$ vertices which, together with B and a $k^{O(f(1/\varepsilon))}$ sized subset of C cover a $(2 + \varepsilon)$ -approximate solution. Finally, the remaining vertices are discarded in an appropriate manner once the relevant information they hold is compiled into a polynomially bounded data structure. This high level approach of identifying “hitters” and “connectors” among the vertices is a natural first step for problems with this flavor [11, 12, 30] and the more involved problem-specific part in each case resides in (a) computing such a partition and (b) setting up a procedure to identify and remove redundant parts of the input, leaving only $k^{O(f(1/\varepsilon))}$ vertices or edges.

Fix $\rho = 2^{O(1/\varepsilon)}$ to be a constant such that $\frac{1}{\lceil \log_2 \rho \rceil} \leq \varepsilon$. Our starting point is a lemma of Fomin et al. [17] showing that there is a polynomial-time algorithm that takes as input the pair (G, k) and either correctly concludes that G has no \mathcal{F} -deletion set (a set of vertices whose deletion leaves an \mathcal{F} -minor free graph) of size at most k or outputs disjoint sets $X, Z \subseteq V(G)$ such that (a) $|X \cup Z| = k^{O(1)}$, (b) X is an \mathcal{F} -deletion set of G , (c) for every connected component C of $G - (X \cup Z)$, $|N(C) \cap Z| \leq 2(\eta + 1)$, and (d) for every $x, y \in X$, either Z intersects every x, y path in $G - (X \setminus \{x, y\})$ or there is an x - y flow of value at least $3k + \eta + 3$ in the graph $G - (X \setminus \{x, y\})$. Let G_τ denote the graph $G - (X \cup Z)$. In the proposed partition (A, B, C) , our intention is to set $B = X \cup Z$.

Given the sets X and Z described above we will compute a subset of vertices covering a $(2 + \varepsilon)$ -approximate connected \mathcal{F} -deletion set (if one of size at most k exists) in two stages. In the first stage, we mark a set of $k^{O(\rho)}$ vertices of G_τ such that for every subset U of $X \cup Z$, if

there is a set T_U of vertices in G_τ using which the vertices in U can be connected, then there is a set of roughly $(1 + \varepsilon)|U|$ *marked* vertices which can do the same connecting task as T_U with respect to the vertices in U . In the second stage, we ignore the connectivity requirement on the subset of the solution in $X \cup Z$ and for each connected component of G_τ , mark $k^{\mathcal{O}(\rho)}$ vertices such that if there is a solution which intersects a component of G_τ then there is a way to select a sufficiently small subset of the *marked* vertices in the component which can be connected to the vertices of the solution in $X \cup Z$ and achieve the same “ \mathcal{F} -minor hitting” behaviour as the original set of vertices.

A major obstacle here is the fact that the number of connected components of G_τ may be unbounded, which might mean that we mark a set of vertices whose size is not bounded by a function of the parameter k at all. It is to overcome this obstacle that we distribute the set $V(G) \setminus B$ among the remaining partitions A and C . In other words, we will be able to partition the components of G_τ into two sets that we call **Type 1** components (corresponding to C) and **Type 2** components (corresponding to A) and then show that the **Type 1** components although unbounded in number, only provide connectivity to a minimal \mathcal{F} -deletion set contained in the solution while, on the other hand, the **Type 2** components are more complex but bounded polynomially (in k) in number. We then argue that there is a way to achieve the objective of our initial marking strategy by marking only a *polynomial* number of vertices of G_τ in total, and these vertices cover a factor- $(2 + \varepsilon)$ approximate solution and moreover, a factor- $\text{OPT}^{\mathcal{O}(1)}$ approximate solution can be recovered by a straightforward examination of the connected components of the subgraph induced by the marked vertices.

To extend this to our approximate compression, we prove the following lemma.

► **Lemma 16.** *There is an algorithm that given G, k, X, Z as described above, runs in time $k^{\mathcal{O}(\rho)} n^{\mathcal{O}(1)}$ and either correctly concludes that G has no connected \mathcal{F} -deletion set of size at most k or returns a set $L \subseteq V(G)$ such that the following statements hold.*

1. $L \supseteq (X \cup Z)$, $|L| = k^{\mathcal{O}(\rho)}$.
2. For every S which is a minimal connected \mathcal{F} -deletion set of G of size at most k , G has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ contained in L .
3. For every connected component C of $G - L$, $|N(C) \cap (L \setminus X)| \leq 2(\eta + 1)$.
4. For every \mathcal{F} -deletion set S of G of size at most $3k$ and every $C \in \mathcal{CC}(G - L)$, $|(N(C) \cap X) \setminus S| \leq \eta + 1$.

The above lemma is obtained following the previously discussed marking steps that led to the factor- $\text{OPT}^{\mathcal{O}(1)}$ approximation and can be seen as an “approximate-solution-capturing” variant of the lemma of Fomin et al. [17] in the context of our problem. Once we have this lemma in hand, we define an annotated version of CONNECTED PLANAR \mathcal{F} -DELETION and encode the output of the lemma above as an instance of ANNOTATED CONNECTED PLANAR \mathcal{F} -DELETION whose size is bounded polynomially in k , which completes the compression, giving us Theorem 1.

4 The $(2 + \varepsilon)$ -Approximate Compression for CONNECTED PLANAR \mathcal{F} -DELETION

Recall that we have chosen $\rho = 2^{\mathcal{O}(1/\varepsilon)}$ to be a constant such that $\frac{1}{\lfloor \log_2 \rho \rfloor} \leq \varepsilon$. Throughout this section, we suppress the dependence of the running time and compression size on \mathcal{F} in the $\mathcal{O}(\cdot)$ notation. However, we make the dependencies on ε explicit. Our first aim in this section is to prove Lemma 17, which is then invoked in the proof of Lemma 16. This lemma says that there is a polynomial-time algorithm that identifies a set of $k^{\mathcal{O}(\rho)}$ vertices that cover a $(2 + \varepsilon)$ -approximate solution (if one of size at most k exists).

► **Lemma 17.** *There is an algorithm that, given G and k , runs in time $k^{\mathcal{O}(\rho)} n^{\mathcal{O}(1)}$ and either correctly concludes that G has no connected \mathcal{F} -deletion set of size at most k or returns a set $V^\infty \subseteq V(G)$ such that the following statements hold.*

1. $|V(G) \setminus V^\infty| = k^{\mathcal{O}(\rho)}$.
2. *For every S which is a minimal connected \mathcal{F} -deletion set of G of size at most k , G has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ that is disjoint from V^∞ .*

Let η be a constant depending only on \mathcal{F} such that every graph which is \mathcal{F} -minor free has treewidth at most η . By Proposition 14, we know that such a constant exists. Let $h = 10\eta + \max_{F \in \mathcal{F}} |F|$. Note that h is an upper bound on the number of vertices and the number of edges in any graph in \mathcal{F} .

We begin building towards the proof of Lemma 17 by recalling the following lemma from [17] and some associated observations. The numbering of the lemmas in our citations is derived from the full version of [16]. A set $S \subseteq V(G)$ is called an \mathcal{F} -deletion set of G if $G - S$ is an \mathcal{F} -minor free graph.

► **Lemma 18** (Lemma 25, [16]). *There is a randomized polynomial-time algorithm that given (G, k) , either concludes that G has no \mathcal{F} -deletion set of size at most k or outputs disjoint sets $X, Z \subseteq V(G)$ satisfying the following properties.*

1. $|X \cup Z| = k^{\mathcal{O}(1)}$.
2. X is an \mathcal{F} -deletion set of G .
3. *For every connected component C of $G - (X \cup Z)$, $|N(C) \cap Z| \leq 2(\eta + 1)$.*
4. *For every $x, y \in X$, either Z intersects every x - y path in $G - (X \setminus \{x, y\})$ or there is an x - y flow of value at least $3k + \eta + 3$ in the graph $G - (X \setminus \{x, y\})$.*

If G has an \mathcal{F} -DELETION set of size k , then this algorithm outputs the pair (X, Z) with probability $(1 - \frac{1}{2^n})$. Otherwise, the algorithm always correctly concludes that no such set exists.

When the pair $\tau = (X, Z)$ is clear from the context, we use G_τ to refer to the graph $G - (X \cup Z)$.

The randomization in Fomin et al.'s proof of this lemma arises due to the execution of a randomized linear-time constant-factor approximation algorithm for (unconnected) PLANAR \mathcal{F} -DELETION, i.e., the subroutine used to compute the set X . However, this step can be replaced with any deterministic factor- $\text{OPT}^{\mathcal{O}(1)}$ approximation for the same problem with, again, only a constant-factor change in the degree of the polynomial in the first property of the lemma. In particular, we can use the factor- $O(\log^{3/2} \text{OPT})$ approximation from [15] and avoid the randomization.

We now recall the following useful properties of non-solution vertices in the neighborhood of any connected component of G_τ .

► **Proposition 19** (Lemma 26, [16]). *Let P be a set of vertices such that for every distinct $u, v \in P$, there is a u - v flow of value $3k + \eta + 3$, let S be an \mathcal{F} -deletion set of G of size at most $3k$ which is disjoint from P and let (T, χ) be a tree decomposition of $G - S$ of width at most η . Then, for every pair of vertices $u, v \in P$, there is a vertex $x \in V(T)$ such that $u, v \in \chi(x)$ and moreover, there is a vertex $y \in V(T)$ such that $P \subseteq \chi(y)$.*

The above proposition captures the fact that if P is disjoint from S and is a set of vertices with pairwise-high flow, then these vertices must appear together in some bag of the tree decomposition (T, χ) . Indeed, if this were not the case, then a pair of vertices in this set can be separated by deleting a small separator in the graph, contradicting the high flow between them. The next statement follows as a consequence of Proposition 19.

► **Proposition 20** (Lemma 26, [16]). *Let G, k and $\tau = (X, Z)$ be as in Lemma 18. For every \mathcal{F} -deletion set S of G of size at most $3k$ and every connected component C of G_τ , $|N(C) \cap X \setminus S| \leq \eta + 1$.*

Indeed, the fourth property ensured by Lemma 18 guarantees that for every connected component C of G_τ , $N(C) \cap X$ is a set of vertices with a pairwise flow of value at least $3k + \eta + 3$ in G . If it were the case that $|N(C) \cap X \setminus S| > \eta + 1$, then Proposition 19 would imply that some bag of a tree decomposition of $G - S$ of width η contains the set $(N(C) \cap X) \setminus S$, which has size greater than $\eta + 1$, a contradiction.

Proposition 20 guarantees that for every connected component C of G_τ , all but at most $\eta + 1$ vertices in $N(C) \cap X$ must be deleted (equivalently, contained in S). Combining this with the third property ensured by Lemma 18, we may conclude that all but at most $3(\eta + 1)$ vertices in $N(C)$ must be contained in S .

Our marking rules rely on the following result of Du et al. [10] which has found applications in other algorithms for connected deletion problems [11, 30].

► **Proposition 21** ([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 $(1 + \frac{1}{\lfloor \log_2 p \rfloor}) \cdot w(T)$.*

4.1 The $\text{OPT}^{\mathcal{O}(1)}$ -approximation for CONNECTED PLANAR \mathcal{F} -DELETION

In what follows, we fix G, k , and $\tau^* = (X, Z)$ as given by Lemma 18.

► **Definition 22.** *For a vertex set C disjoint from X , a set $J \subseteq N(C)$ is called a set compatible with C if $|J \cap X| \leq \eta + 1$. We denote by \mathcal{B}_C the set of all sets compatible with C . For a set C and set J compatible with C , the $|J|$ -boundaried graph $(G[C \cup J], J)$ is denoted by $G_{C,J}$.*

Recall that $\forall C \in \mathcal{CC}(G_{\tau^*})$, $|N(C) \cap Z| \leq 2(\eta + 1)$ (argued using the third property of Lemma 18). Consequently, for every $C \in \mathcal{CC}(G_{\tau^*})$, any set $J \subseteq N(C)$ compatible with C has size at most $3\eta + 3$, which is a constant in our setting. Moreover, due to Proposition 20, we have that for any solution S , the set $N(C) \setminus S$ must be a set compatible with C .

► **Lemma 23.** *There is an algorithm that given G, k , and $\tau^* = (X, Z)$, runs in polynomial time and returns a set $P^\infty \subseteq V(G) \setminus (X \cup Z)$ and a partition of $\mathcal{CC}(G_{\tau^*})$ into sets \mathcal{P} and \mathcal{Q} such that the following statements hold.*

1. $|\mathcal{P}| = k^{\mathcal{O}(1)}$ and $P^\infty \subseteq V(\mathcal{P})$.
2. Every \mathcal{F} -deletion set of $G - V(\mathcal{Q}) - P^\infty$ of size at most $3k$ is an \mathcal{F} -deletion set of G .

► **Definition 24.** *Let $G, k, \tau^* = (X, Z), \mathcal{P}, \mathcal{Q}$ be as in Lemma 23. We call every component in \mathcal{Q} a Type 1 component of G_τ and every component in \mathcal{P} a Type 2 component of G_{τ^*} .*

Lemma 23 implies that the vertices contained in Type 1 components are only required for providing connectivity between those vertices of a minimal \mathcal{F} -deletion set of size at most $3k$ that are contained in $X \cup Z$ and the Type 2 components are bounded polynomially in k . The proof of this lemma closely follows the proof of Fomin et al. (see Lemma 36, [16]) with the following difference: instead of discarding irrelevant vertices, we identify them and use them to define the sets \mathcal{P} and \mathcal{Q} .

We are now ready to state our marking rules. We fix $G, k, \tau^* = (X, Z), P^\infty, \mathcal{P}, \mathcal{Q}$ as given by Lemma 23.

► **Marking Rule 1.** For every $R \subseteq X \cup Z$ of size at most ρ , we compute an optimal R -Steiner tree T_R in G with the unitary cost function (denoted w_1). If $|V(T_R)| \leq (2 + \varepsilon)k$, then we mark the vertices of T_R .

Before we describe our next marking rule, we need the following definition.

► **Definition 25.** For a graph G and vertex $p \in V(G)$, we say that a set $S \subseteq V(G)$ is a p -connected set in G if $p \notin S$ and $G[S \cup \{p\}]$ is connected. For a t -boundaried graph (G, L) , $P \subseteq V(G) \setminus L$, $k \in \mathbb{N}$ and $p \in V(G) \setminus L$, a set $S \subseteq V(G)$ is called a (p, P, k) -representative set for (G, L) if the following holds:

For every p -connected set $S \subseteq V(G) \setminus L$ of size at most k in G , S contains a p -connected set $\tilde{S} \subseteq V(G) \setminus L$ of size at most $|S|$ such that $\mathcal{M}_h(G - p - (S \cup P), L) \supseteq \mathcal{M}_h(G - p - (\tilde{S} \cup P), L)$.

We extend the notion of p -connected sets in a natural way to the additional case when p is not a vertex, but $p = \emptyset$. In this case, S is p -connected if $G[S]$ is connected. The definition of (p, P, k) -representative sets for $p = \emptyset$ is analogous.

The notion of p -connected sets is motivated by the following observation.

► **Observation 26.** Let S be a connected \mathcal{F} -deletion set of G of size at most $3k$ and let $C \in \mathcal{CC}(G_\tau)$. Let $S_C = S \cap C$ and suppose that S_C is non-empty. Let $J = N(C) \setminus S$ and consider the boundaried graph (G_C^J, J) , where G_C^J is defined as the graph obtained from $G[N[C]]$ as follows. If $T_C = N(C) \cap S$ is non-empty, then we identify the vertices in T_C into a single vertex v_J^* to obtain G_C^J . Otherwise, $G_C^J = G[N[C]]$ and set $v_J^* = \emptyset$. Then, S_C is a v_J^* -connected set of size at most $3k$ in the graph G_C^J .

The definition of the (p, P, k) -representative set S (Definition 25) guarantees that for every p -connected set $S \subseteq V(G) \setminus L$ of size at most k in G , there is a p -connected set in S that provides the “same” connectivity as S with respect to p , costs at most $|S|$, and hits every minor model hit by S in $G - p - P$ (and possibly more). The case when $p = \emptyset$ covers the case when the entire solution is contained in a single component of G_{τ^*} .

► **Lemma 27.** Let (G, L) be a t -boundaried graph for some $t \leq 3\eta + 3$, $k \in \mathbb{N}$, $p \in V(G) \setminus L$ and $P \subseteq V(G) \setminus L$. Then, there is a (p, P, k) -representative set for (G, L) of size $\mathcal{O}(k)$. Moreover, if $\text{tw}(G) = \mathcal{O}(\eta)$, then such a set can be computed in polynomial time.

We are now ready to describe our second and final marking rule. Recall that \mathcal{P} , \mathcal{Q} and P^∞ are as given by Lemma 23. Let C be a Type 2 component of $\mathcal{CC}(G_{\tau^*})$. Then, $C \in \mathcal{P}$. Let P_C^∞ denote the set $P^\infty \cap C$.

► **Marking Rule 2.** For each compatible set $J \in \mathcal{B}_C$, we construct a $|J|$ -boundaried graph (G_C^J, J) defined as the graph obtained from $(G[N[C]], J)$ by identifying all vertices of $N(C) \setminus J$ with the resulting vertex denoted by v_J^* . If $J = N(C)$, then we simply define $v_J^* := \emptyset$. We then use Lemma 27 to compute a $(v_J^*, P_C^\infty, 3k)$ -representative set Q_C^J of size $\mathcal{O}(k)$ for (G_C^J, J) and mark the vertices in it.

In the rule above, J is intended to represent the subset of $N(C)$ that is *not* deleted by a hypothetical connected \mathcal{F} -deletion set of size at most $3k$ that intersects $N[C]$. If $N(C) = J$, then any hypothetical connected \mathcal{F} -deletion set that intersects $N[C]$ and has an empty intersection with $N(C)$ must also be contained entirely in C . Consequently, we simulate this case by setting $v_J^* = \emptyset$. Note that (G_C^J, J) is a t -boundaried graph for some $t \leq 3\eta + 3$ because $|J \cap X| \leq \eta + 1$ and $|J \cap Z| \leq 2(\eta + 1)$. Moreover, $\text{tw}(G_C^J) \leq 2\eta + 2$. As the treewidth of $G[C \cup (N(C) \cap Z)]$ is at most η , it follows that adding v_J^* and a subset of X of size at most $\eta + 1$ can increase the treewidth of $G[C \cup (N(C) \cap Z)]$ to at most $2\eta + 2$.

We are now ready to prove Lemma 17.

Proof of Lemma 17. We first invoke Lemma 18 to either correctly conclude that G has no \mathcal{F} -deletion set of size at most k or compute the pair $\tau^* = (X, Z)$. In the former case we return the same. Otherwise, we invoke Lemma 23 to compute the partition $\mathcal{P} \uplus \mathcal{Q}$ of $\mathcal{CC}(G_{\tau^*})$ and the set $P^\infty \subseteq V(\mathcal{P})$. Recall that the components contained in \mathcal{P} are called Type 2 components and those contained in \mathcal{Q} are called Type 1 components. We now execute Marking Rule 1.

For each $C \in \mathcal{CC}(G_{\tau^*})$, let R_C denote the set of marked vertices in the connected component C . Note that the *total* number of vertices marked across all components of G_{τ^*} in this step is at most $3k \cdot |X \cup Z|^\rho = k^{\mathcal{O}(\rho)}$.

Now, for each Type 2 component $C \in \mathcal{CC}(G_{\tau^*})$ we execute Marking Rule 2 with Q_C denoting the subset of its vertices marked in this execution. For every Type 1 component C , $Q_C = \emptyset$. Note that the number of vertices marked in each Type 2 component C in this step is bounded by $\mathcal{O}(k \cdot |\mathcal{B}_C|) = k^{\mathcal{O}(1)}$. Since the number of Type 2 components is bounded by $k^{\mathcal{O}(1)}$, we conclude that Marking Rule 2 marks a total of $k^{\mathcal{O}(1)}$ vertices across all connected components of G_{τ^*} .

We now argue that we have preserved a $(2 + \varepsilon)$ -approximation in the union of the marked set of vertices and $X \cup Z$.

▷ **Claim 28.** If G has a connected \mathcal{F} -deletion set S of size at most k , then it has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ contained in $X \cup Z \cup \bigcup_{C \in \mathcal{CC}(G_{\tau^*})} (Q_C \cup R_C)$.

Proof sketch. Let $S'' = S \cap (X \cup Z)$ and $A = S \setminus S''$. Suppose that $G[S'']$ is not connected. Consider the terminal set S'' with a weight function on the edges that assigns 0 to every edge with both endpoints in S'' and assigns 1 to every other edge. Then, we observe that $G[S]$ contains an S'' -Steiner tree with weight at most $|A| + \beta - 1$ where β is the number of connected components in $G[S'']$ and Proposition 21 guarantees that the set of vertices marked by Marking Rule 1 contains a set A'' such that $G[A'' \cup S'']$ is connected, $|A''| + \beta - 1 \leq (1 + \varepsilon)(|A| + \beta - 1)$, implying that $|A'' \cup S''| \leq (1 + \varepsilon)|S|$.

In order to prove the claim, we will show that once we have added A'' to S'' , we can (a) ignore all other vertices of Type 1 components originally contained in S and (b) consider each Type 2 component $C \in \mathcal{CC}(G_{\tau^*})$ independently and replace the set $S \cap C$ with a subset of Q_C of size at most $|S \cap C|$ in such a way that the resulting set is eventually still connected and hits all \mathcal{F} -minor models in G . Observe that if $S'' = \emptyset$, then $A'' = \emptyset$. Moreover, if $G[S'']$ is connected, then we set $A'' = \emptyset$. ◁

Finally, we define V^∞ as follows. $V^\infty = V(G) \setminus (X \cup Z \cup \bigcup_{C \in \mathcal{CC}(G_{\tau^*})} (Q_C \cup R_C))$. ◀

Theorem 2 can now be proved using Lemma 17.

4.2 The approximate compression for CONNECTED PLANAR \mathcal{F} -DELETION

The crux of our approximate compression is the following stronger version of Lemma 18, where we have now also ensured the presence of a $(2 + \varepsilon)$ -approximate solution within a polynomially bounded set L (which plays the role that the set $X \cup Z$ plays in Lemma 18).

► **Lemma 29.** *There is an algorithm that given $G, k, \tau^* = (X, Z)$ from Lemma 18, runs in time $k^{\mathcal{O}(\rho)} n^{\mathcal{O}(1)}$ and either correctly concludes that G has no connected \mathcal{F} -deletion set of size at most k or returns a set $L \subseteq V(G)$ such that the following statements hold.*

1. $L \supseteq (X \cup Z)$, $|L| = k^{\mathcal{O}(\rho)}$.
2. For every S which is a minimal connected \mathcal{F} -deletion set of G of size at most k , G has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ contained in L .

3. For every connected component C of $G - L$, $|N(C) \cap (L \setminus X)| \leq 2(\eta + 1)$.
4. For every \mathcal{F} -deletion set S of G of size at most $3k$ and any $C \in \mathcal{CC}(G - L)$, $|(N(C) \cap X) \setminus S| \leq \eta + 1$.

Proof. We invoke Lemma 17 to either conclude that G has no \mathcal{F} -deletion set of size at most k or compute the set V^∞ such that $|V(G) \setminus V^\infty| = k^{\mathcal{O}(\rho)}$ and for every S which is a minimal connected \mathcal{F} -deletion set of G of size at most k , G has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ disjoint from V^∞ . In the former case we return the same. Otherwise, let $Y = V(G) \setminus V^\infty$. Although it does not follow from the statement of Lemma 17, an inspection of the definition of V^∞ in the proof shows that $V^\infty \cap (X \cup Z) = \emptyset$ and so we may assume without loss of generality that $Y \supseteq X \cup Z$.

Consider the graph $G - X$ which is known to be \mathcal{F} -minor free and hence has treewidth at most η . Let (T, χ) be an arbitrary tree decomposition of $G - X$ of minimum width and suppose that it is arbitrarily rooted. We now define L to be the set $X \cup \text{LCA-closure}(T, \chi, Y \setminus X)$. The fact that $L \supseteq X \cup Z$ and $|L| = k^{\mathcal{O}(\rho)}$ follows from the fact that $X \cup Z \subseteq Y \subseteq L$, $|Y| = k^{\mathcal{O}(\rho)}$, and Lemma 10. Moreover, Lemma 10 also implies that the third statement holds. Similarly, the second statement follows from the fact that for every S which is a minimal connected \mathcal{F} -deletion set of G of size at most k , G has a connected \mathcal{F} -deletion set of size at most $(2 + \varepsilon)|S|$ contained in Y which is contained in L .

For the final statement, we know from Lemma 18 that for every $x, y \in X$, either $Z \subseteq Y \subseteq L$ intersects every x, y path in $G - (X \setminus \{x, y\})$ or there is an x - y flow of value at least $3k + \eta + 3$ in the graph $G - (X \setminus \{x, y\})$. Consequently, we can invoke Proposition 20 with the pair $(X, L \setminus X)$ instead of (X, Z) to conclude that for every \mathcal{F} -deletion set S of G of size at most $3k$ and any $C \in \mathcal{CC}(G - L)$, $|(N(C) \cap X) \setminus S| \leq \eta + 1$. This completes the proof of the lemma. \blacktriangleleft

Before proceeding to the description of the compression (Theorem 1), we need to define an annotated version of our problem. Recall that $h \geq \max\{10\eta, \max_{F \in \mathcal{F}} |F|\}$. In the ANNOTATED CONNECTED \mathcal{F} -DELETION problem, the input is a graph G , integer k , and a set $\mathcal{R} = \{(P_i, Q_i, \mathcal{T}_i)\}_{i \in [\ell]}$ where $\forall i \in [\ell]$, $P_i \cap Q_i = \emptyset$, $P_i \cup Q_i \subseteq V(G)$, $|Q_i| \leq 3(\eta + 1)$, and \mathcal{T}_i is a set of $|Q_i|$ -labeled graphs of size at most h . The goal is to decide whether there is a set $S \subseteq V(G)$ of size at most k such that $G[S]$ is connected and G_S has no minor isomorphic to a graph in \mathcal{F} , where G_S is defined as the graph obtained from G by going over all $i \in [\ell]$ and gluing a graph (H_i, Q_i) on to (G, Q_i) using the canonical labeling, whenever $P_i \subseteq S$ and $Q_i \cap S = \emptyset$, and where $\mathcal{M}_h(H_i, Q_i) = \mathcal{T}_i$.

A set S (not necessarily of size at most k) satisfying the properties above is said to be a feasible solution. Following the convention from [26], the parameterized optimization version of ANNOTATED CONNECTED \mathcal{F} -DELETION is defined to be a minimization problem with the optimization function $ACFD : \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\infty\}$ defined as follows.

$$ACFD(G, \mathcal{R}, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a feasible solution,} \\ \min\{|S|, k + 1\} & \text{otherwise.} \end{cases}$$

We next describe a compression algorithm that takes as input an instance of CONNECTED \mathcal{F} -DELETION and outputs an instance of ANNOTATED CONNECTED \mathcal{F} -DELETION with size bounded polynomially in k and in which a $(2 + \varepsilon)$ -approximate solution for the original instance is preserved.

► **Lemma 30.** *There is an algorithm that given G, k , runs in time $k^{\mathcal{O}(\rho)} n^{\mathcal{O}(1)}$ and either correctly concludes that G has no connected \mathcal{F} -deletion set of size at most k or returns an instance $\Gamma = (\tilde{G}, \tilde{k}, \tilde{\mathcal{R}} = \{(\tilde{P}_i, \tilde{Q}_i, \tilde{\mathcal{T}}_i)\}_{i \in [\ell]})$ of ANNOTATED CONNECTED \mathcal{F} -DELETION such that the following statements hold.*

1. $|V(\tilde{G})| = k^{\mathcal{O}(\rho)}$, $\ell = k^{\mathcal{O}(\rho)}$.
2. If G has a connected \mathcal{F} -deletion set S of size at most k , then Γ has a feasible solution of size at most $(2 + \varepsilon)|S|$.
3. Every feasible solution of Γ is a connected \mathcal{F} -deletion set of G .

Proof sketch. We begin by executing Lemma 18 and then Lemma 29 to either conclude that G has no connected \mathcal{F} -deletion set of size at most k or compute $\tau = (X, Z)$ and the set $L \supseteq X \cup Z$. We set $\tilde{G} = G[L]$ and $\tilde{k} = (2 + \varepsilon)k$. Recall that $|L| = k^{\mathcal{O}(\rho)}$.

If the number of connected components of $G - L$ is not already bounded by $k^{\mathcal{O}(\rho)}$, then we use the arguments from Lemma 23 (with L used in place of $X \cup Z$) to partition them into \mathcal{P} and \mathcal{Q} and compute the set $P^\infty \subseteq V(\mathcal{P})$ such that every \mathcal{F} -deletion set of size at most $3k$ for $G - V(\mathcal{Q}) - P^\infty$ is an \mathcal{F} -deletion set of G and $|\mathcal{P}| = k^{\mathcal{O}(\rho)}$. Observe that because we are now using L in place of $X \cup Z$, the size of the set $|\mathcal{P}|$ is now bounded by $k^{\mathcal{O}(\rho)}$ (because $|L| = k^{\mathcal{O}(\rho)}$) as opposed to $k^{\mathcal{O}(1)}$ where the degree of the polynomial only depends on \mathcal{F} .

We now define $\tilde{\mathcal{R}}$ as follows. Let $\hat{G} = G - V(\mathcal{Q}) - P^\infty$. For every $C \in \mathcal{CC}(\hat{G})$ and compatible set $J \in \mathcal{B}_C$ in the graph \hat{G} , we add to $\tilde{\mathcal{R}}$ the tuple $(N(C) \setminus J, J, \mathcal{M}_h(\hat{G}_C^J, J))$. This completes the definition of $\tilde{\mathcal{R}}$ and consequently, the definition of Γ . We now show that the three properties specified in the lemma hold.

Since we have $k^{\mathcal{O}(\rho)}$ connected components in $\mathcal{CC}(\hat{G})$ and each connected component has $k^{\mathcal{O}(1)}$ compatible sets, we conclude that $\ell = k^{\mathcal{O}(\rho)}$. It remains to prove the second and third statements of the lemma. For the second statement, Lemma 29 guarantees that if G has a connected \mathcal{F} -deletion set S of size at most k then it has a connected \mathcal{F} -deletion set S' of size at most $(2 + \varepsilon)|S|$ which is contained in L . We claim that S' is also a feasible solution for the constructed instance of ANNOTATED CONNECTED \mathcal{F} -DELETION. If this were not the case then there is an \mathcal{F} -minor model in the graph $\tilde{G}_{S'}$ as defined in the definition of the problem. However, the definition of the tuples in $\tilde{\mathcal{P}}$ implies that every \mathcal{F} -minor model present in $\tilde{G}_{S'}$ is also present in $G - S'$, a contradiction to S' being a connected \mathcal{F} -deletion set of G . Therefore, we conclude that the second statement holds.

For the third statement, let S be a feasible solution for the constructed instance of ANNOTATED CONNECTED \mathcal{F} -DELETION and suppose that it is not a connected \mathcal{F} -deletion set of G . Then, there is an \mathcal{F} -minor model in $G - S$. In fact, due to Lemma 23 and the definition of the sets P^∞ and \mathcal{Q} , we conclude that there is an \mathcal{F} -minor model in $G - V(\mathcal{Q}) - P^\infty - S = \hat{G} - S$ because otherwise S must be an \mathcal{F} -deletion set of G as well.

However, for every \mathcal{F} -minor model in $\hat{G} - S$, one can invoke “cut and paste” arguments similar to those used in the proof of Lemma 23 to construct an alternate minor model in \tilde{G}_S for the same graph in \mathcal{F} , a contradiction to our choice of S as a feasible solution of Γ . This completes the proof of the lemma. \blacktriangleleft

Theorem 1 now follows from Lemma 30. The reduction algorithm uses the algorithm of this lemma and the solution-lifting algorithm simply returns the approximate solution computed for the instance of ANNOTATED CONNECTED \mathcal{F} -DELETION.

5 Conclusion and Open Problems

Our result on the approximate compressibility of the connectivity constrained variant of PLANAR \mathcal{F} -DELETION demonstrates that preprocessing lower bounds can be side-stepped even for very general versions of problems such as Vertex Cover and Feedback Vertex Set as long as one allows a small loss in accuracy. While a step forward in our understanding of preprocessing under connectivity constraints, our work leaves some natural questions for follow-up work.

1. Is there a $(1 + \varepsilon)$ -approximate kernel for this problem (for every $0 < \varepsilon < 1$)?
2. What is the best approximation factor one can achieve for this problem in polynomial time?
3. Is there a fixed-parameter algorithm for CONNECTED PLANAR \mathcal{F} -DELETION with a single-exponential dependence on k ? To the best of our knowledge, such an algorithm is not known in the literature even when $\mathcal{F} = \{\theta_c\}$ for $c \geq 3$. Here, θ_c is the graph on 2 vertices with c parallel edges between them.

References

- 1 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.
- 2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 4 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.
- 5 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.
- 6 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.
- 7 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
- 8 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.
- 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 Samuel Fiorini, Gwenaél Joret, and Ugo Pietropaoli. Hitting diamonds and growing cacti. In *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, pages 191–204, 2010. doi:10.1007/978-3-642-13036-6_15.
- 15 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discrete Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.

- 16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar \mathcal{F} -deletion: Approximation, kernelization and optimal FPT algorithms.
- 17 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.
- 18 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.
- 19 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.
- 20 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015.
- 21 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.
- 22 Gwenaél Joret, Christophe Paul, Ignasi Sau, Saket Saurabh, and Stéphan Thomassé. Hitting and harvesting pumpkins. *SIAM J. Discrete Math.*, 28(3):1363–1390, 2014. doi:10.1137/120883736.
- 23 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.
- 24 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.
- 25 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.
- 26 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, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017. doi:10.1145/3055399.3055456.
- 27 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.
- 28 Jesper Nederlof. Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 713–725, 2009. doi:10.1007/978-3-642-02927-1_59.
- 29 Geevarghese Philip, Venkatesh Raman, and Yngve Villanger. A quartic kernel for pathwidth-one vertex deletion. In *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, pages 196–207, 2010. doi:10.1007/978-3-642-16926-7_19.
- 30 M. S. Ramanujan. An approximate kernel for connected feedback vertex set. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 77:1–77:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.77.

Restricted Adaptivity in Stochastic Scheduling

Guillaume Sagnol 

Institut für Mathematik, TU Berlin, Germany

Daniel Schmidt genannt Waldschmidt 

Institut für Mathematik, TU Berlin, Germany

Abstract

We consider the stochastic scheduling problem of minimizing the expected makespan on m parallel identical machines. While the (adaptive) list scheduling policy achieves an approximation ratio of 2, any (non-adaptive) fixed assignment policy has performance guarantee $\Omega\left(\frac{\log m}{\log \log m}\right)$. Although the performance of the latter class of policies are worse, there are applications in which non-adaptive policies are desired. In this work, we introduce the two classes of δ -delay and τ -shift policies whose degree of adaptivity can be controlled by a parameter. We present a policy – belonging to both classes – which is an $\mathcal{O}(\log \log m)$ -approximation for reasonably bounded parameters. In other words, an exponential improvement on the performance of any fixed assignment policy can be achieved when allowing a small degree of adaptivity. Moreover, we provide a matching lower bound for any δ -delay and τ -shift policy when both parameters, respectively, are in the order of the expected makespan of an optimal non-anticipatory policy.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases stochastic scheduling, makespan minimization, approximation algorithm, fixed assignment policy, non-anticipatory policy

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.79

Related Version *Full Version:* <https://arxiv.org/abs/2106.15393> [30]

Funding Guillaume Sagnol and Daniel Schmidt genannt Waldschmidt were funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

Acknowledgements We thank Thibault Jaillard for helpful discussions on the topic of this paper. We also thank the anonymous referees for helpful comments.

1 Introduction

Load balancing problems are one of the most fundamental problems in the field of scheduling, with applications in various sectors such as manufacturing, construction, communication or operating systems. The common challenge is the search for an efficient allocation of scarce resources to a number of tasks. While many variants of the problem are already hard to solve, in addition one may have to face uncertainty regarding the duration of the tasks; one way to model this is to use stochastic information learned from the available data.

In contrast to the solution concept of a schedule in deterministic problems, we are concerned with *non-anticipatory policies* in stochastic scheduling problems. Such a policy has the ability to *react* to the information observed so far. While this adaptivity can be very powerful, there are situations where assigning resources to jobs prior to their execution is a highly desired feature, e.g. for the scheduling of healthcare services. This is especially true for the daily planning of elective surgery units in hospitals, where a sequence of patients is typically set in advance for each operating room. In this work, we present and analyze semi-adaptive policies, which allow one to control the level of adaptivity of the policy.



© Guillaume Sagnol and Daniel Schmidt genannt Waldschmidt;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 79; pp. 79:1–79:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Snippets of the execution of a δ -delay policy: Realizations of jobs observed up to time t (left) and up to some time $> t + \delta$ (right) are depicted by rectangles in dark grey; the running job non-completed by the time of each snippet is indicated by squared dots in dark grey; jobs that did not start yet are depicted in light grey with the corresponding machine assignment.

The problem considered in this paper is the stochastic counterpart of the problem of minimizing the makespan on parallel identical machines, denoted by $P \parallel \mathbb{E}[C_{\max}]$ using the three field notation due to Graham, Lawler, Lenstra and Rinnooy Kan [13]. The input consists of a set of n jobs \mathcal{J} and a set of m parallel identical machines \mathcal{M} . Each job $j \in \mathcal{J}$ is associated with a non-negative random variable P_j representing the processing time of the job. The processing times are assumed to be (mutually) independent and to have finite expectation. In this work, it is sufficient to only know the expected processing times.

Roughly speaking, a non-anticipatory policy may, at any point in time t , decide to start a job on an idle machine or to wait until a later decision time. However, it may not anticipate any future information of the realizations, i.e., it may only make decisions based on the information observed up to time t . For further details we refer to the work by Möhring, Radermacher and Weiss [26]. The task is to find a non-anticipatory policy minimizing the expected makespan $\mathbb{E}[C_{\max}] := \mathbb{E}[\max_{j \in \mathcal{J}} C_j]$, where C_j denotes the (random) completion time of job j under the considered policy. An optimal policy is denoted by OPT. By slight abuse of notation we use Π for both the policy and the expected makespan of the policy.

An alternative way of understanding non-anticipatory policies is that they maintain a queue of jobs for every machine. At any point in time t it may start the first job in the queue of a machine if it is idle or it may change the queues arbitrarily, using only the information observed up to time t . In this work we consider a policy to be adaptive if it has the ability to react to the observations by changing the queues arbitrarily. The important class of non-idling non-adaptive policies is called the class of fixed assignment policies. Such a policy assigns all jobs to the machines beforehand, in form of ordered lists, and each machine processes the corresponding jobs as early as possible in this order.

While the class of (fully adaptive) non-anticipatory policies and the class of (non-adaptive) fixed assignment policies can be considered as two extremes, the purpose of this paper is to introduce two classes of policies bridging the gap between them continuously.

► **Definition 1.1** (δ -delay and τ -shift policies). *A δ -delay policy for $\delta > 0$ is a non-anticipatory policy which starts with a fixed assignment of all jobs to the machines and which may, at any point in time t , reassign not-started jobs to other machines with a delay of δ : the reassigned jobs are not allowed to start before time $t + \delta$.*

A τ -shift policy for $\tau > 0$ is a non-anticipatory policy which starts with a fixed assignment of all jobs to the machines and which may reassign jobs to other machines, but only at times that are an integer multiple of τ .

Snippets of the execution of a δ -delay policy and a τ -shift policy can be found in Figure 1 and Figure 2, respectively. Observe that we recover the class of fixed assignment policies by letting δ or τ go to ∞ , and the class of non-anticipatory in the limit when δ or τ goes to 0.

Related Work. Minimizing the makespan on parallel identical machines is a fundamental deterministic scheduling problem which dates back to the 60s. Graham [11] showed that the list scheduling algorithm computes a solution which is within a factor of $(2 - \frac{1}{m})$ away from



■ **Figure 2** Snippets of the execution of a τ -shift policy: Realizations of jobs observed up to time 2τ (left) and some time $> 2\tau$ (right) are depicted by rectangles in dark grey; the running job non-completed by the time of each snippet is indicated by squared dots in dark grey; jobs that did not start yet are depicted in light grey with the corresponding machine assignment.

an optimal solution. When the jobs are arranged in LPT-order, i.e., in non-increasing order of their processing times, he showed that list scheduling gives a $(\frac{4}{3} - \frac{1}{3m})$ -approximation [12]. While $Pm||C_{\max}$, where the number of machines m is constant, and $P||C_{\max}$ are (weakly) and strongly NP-complete [9], respectively, Sahni [32] and Hochbaum and Shmoys [18] obtained a FPTAS and a PTAS, respectively. In subsequent work [3, 5, 17, 21, 22] the running time of the PTAS was improved. More general machine environments were also considered in the literature [19, 24].

The stochastic counterpart $P||E[C_{\max}]$ where the processing times of the jobs are random and the objective is to minimize the expected makespan has also attracted attention. One can easily see that the list scheduling algorithm by Graham [11] also yields a 2-approximation compared to an optimal non-anticipatory policy for the stochastic problem, as its analysis can be carried over to any realization. While list scheduling can be considered as a very adaptive policy, some applications require rather restricted policies, e.g. when scheduling operating rooms at a hospital [7, 36]. A class of non-adaptive policies analyzed in the literature is comprised of fixed assignment policies, in which jobs must be assigned to the machines beforehand. Although more applicable, it is well known that the performance guarantee of an optimal fixed assignment is at least of the order $\Omega\left(\frac{\log m}{\log \log m}\right)$ with respect to an optimal non-anticipatory policy; see [14]. Much work was done in designing fixed assignment policies that are within a constant factor of an optimal fixed assignment policy. Kleinberg, Rabani and Tardos [23] obtain a constant factor approximation for this problem for general probability distributions. When the processing times are exponentially and Poisson distributed, PTASes were found [10, 6]. For the more general problem of makespan minimization on unrelated machines, Gupta, Kumar, Nagarajan and Shen [14] obtained a constant factor approximation. Closely related to the makespan objective, Molinaro [28] obtained a constant factor approximation for the ℓ_p -norm objective. In contrast to the literature for minimizing the makespan where approximative results were compared to an optimal fixed assignment policy, much work on the min-sum objective was done for designing approximative policies compared to an optimal non-anticipatory policy [27, 25, 34, 35, 15]. When minimizing the sum of weighted completion times, Skutella, Sviridenko and Uetz [35] showed that the performance ratio of an optimal fixed assignment policy compared to an optimal non-anticipatory policy can be as large as $\Omega(\Delta)$, where Δ is an upper bound on the squared coefficient of variation of the random variables. Lastly, Sagnol, Schmidt genannt Waldschmidt and Tesch [31] considered the extensible bin packing objective, for which they showed that the fixed assignment policy induced by the LEPT order has a tight approximation ratio of $1 + e^{-1}$ with respect to an optimal non-anticipatory policy.

Closely related to the reassignment of jobs in δ -delay and τ -shift policies, various non-preemptive scheduling problems with migration were considered in offline and online settings. Aggarwal, Motwani and Zhu [1] examined the offline problem where one must perform budgeted migration to improve a given schedule. For online makespan minimization on

parallel machines, different variants on limited migration, e.g. bounds on the processing volume [33] or bounds on the number of jobs [2], were studied. Another related online problem was considered by Englert, Ozmen and Westermann [8] where a reordering buffer can be used to defer the assignment of a limited number of jobs.

One source of motivation for this research is the aforementioned application to surgery scheduling. In this domain, a central problem is the allocation of patients to operating rooms. Although additional resource constraints exist, the core of the problem can be modeled as the allocation of jobs with stochastic durations to parallel machines [7]. In this field, committing to a fixed assignment policy is common practice in order to simplify staff management and reduce the stress level in the operating theatre [4, 29, 36]. Another obstacle to the introduction of sophisticated adaptive policies is the reluctance of computer-assisted scheduling systems among practitioners [20]. That being said, it is clear that resource reallocations do occasionally occur in operating rooms to deal with unforeseen events, hence, giving a reason to study some kind of semi-adaptive model. The proposed model of δ -delay is an attempt to take into account the organizational overhead associated with rescheduling decisions; the model of τ -shift policy by the fact that rescheduling decisions cannot be made at any point in time, but must be agreed upon in short meetings between the OR manager and the medical team. Moreover, we point out that the class of τ -shift policies encompasses the popular class of *proactive-reactive policies* used for the more general resource constrained project scheduling problem [16], in which a baseline schedule can be reoptimized after a set of predetermined decision points (these approaches typically consider a penalty in the objective function to account for deviations between the initial baseline schedule and the reoptimized ones).

Our Contribution. We introduce and analyze two new classes of policies (δ -delay and τ -shift policies) that interpolate between the two extremes of non-adaptive and adaptive policies. For the stochastic problem of minimizing the expected makespan on m parallel identical machines, we analyze the policy $\text{LEPT}_{\delta,\alpha}$, which belongs to the intersection of both classes. This policy can in fact be seen as a generalization of the list policy LEPT , which waits for predefined periods of time before reassigning the non-yet started jobs, taking the delay of δ into account. While an optimal fixed assignment policy has performance guarantee of at least $\Omega\left(\frac{\log m}{\log \log m}\right)$ compared to an optimal non-anticipatory policy, we show that $\text{LEPT}_{\delta,\alpha}$ is an $\mathcal{O}(\log \log m)$ -approximation for some constant $\alpha > 0$ and all $\delta = \mathcal{O}(1) \cdot \text{OPT}$. Therefore, we exponentially improve the performance of non-adaptive policies by allowing a small amount of adaptivity. Moreover, we provide a matching lower bound for δ -delay policies as well as for τ -shift policies if δ or τ are in $\Theta(\text{OPT})$. This shows that there is no δ -delay or τ -shift policy beating the approximation ratio of $\text{LEPT}_{\delta,\alpha}$ by more than a constant factor.

Organization. Section 2 is devoted for the upper bound on the performance guarantee of $\text{LEPT}_{\delta,\alpha}$. A lower bound on optimal δ -delay policies as well as τ -shift policies is given in Section 3. At the end, we conclude and give possible future research directions. Useful results from probability theory and detailed proofs can be found in the appendix of the full version of this paper [30].

2 Upper Bound

In this section, we show that there exists $\alpha > 1$ such that the policy $\text{LEPT}_{\delta,\alpha}$ (see Definition 2.5) has a performance guarantee doubly logarithmic in m if $\delta = \mathcal{O}(1) \cdot \text{OPT}$.

► **Theorem 2.1.** *There exists $\alpha > 1$ such that $\text{LEPT}_{\delta,\alpha}$ is an $\mathcal{O}(\log \log m)$ -approximation for $\delta = \kappa \cdot \text{OPT}$ for any constant $\kappa > 0$.*

In the following, we show Theorem 2.1 for $\alpha = 33$. We note that we did not optimize the constants appearing in our calculation as our lower bound shows that $\log \log(m)$ is the correct order. Notice that it suffices to show the performance guarantee for m large enough as for $m = \mathcal{O}(1)$ the trivial policy assigning all jobs to a single machine is a constant factor approximation. To prove the main theorem, we proceed as follows: First, we define and discuss properties of the fixed assignment policy FLEPT as it lies at the heart of our policy called $\text{LEPT}_{\delta,\alpha}$. After we give the formal definition of $\text{LEPT}_{\delta,\alpha}$, we derive lower bounds on OPT needed to show its performance guarantee. The remaining part is devoted to show Theorem 2.1. The main idea of the proof is that the policy works over a sequence of reassignment periods; at the beginning of each period, there is a constant fraction of available machines with high probability. This can be used to show the following squaring effect: if the remaining volume of non-started jobs is $\epsilon \cdot m \cdot \text{OPT}$ in a period, it will be at most $\epsilon^2 \cdot m \cdot \text{OPT}$ in the next period, with high probability.

Recall that the List Scheduling algorithm due to Graham [11] with respect to a list of all jobs schedules the next job in the list on the next idle machine. Let us define the fixed assignment policy induced by list scheduling in LEPT order.

► **Definition 2.2** (The fixed assignment policy FLEPT). *Let all jobs be arranged in non-increasing order of their expected processing times. FLEPT is the fixed assignment policy that assigns the jobs in this order to the same machines as List Scheduling would yield for the deterministic instance in which the processing times are replaced by their expected value.*

As shown by Sagnol, Schmidt genannt Waldschmidt and Tesch [31], FLEPT admits bounds on the expected load of any machine captured in the next lemma.

► **Lemma 2.3** ([31]: Section 3, Lemma 3). *Given an assignment of jobs to machines induced by FLEPT, let ℓ_i denote the expected load of machine i , i.e., the sum of expected processing times of the jobs assigned to i . Moreover, let n_i denote the number of jobs assigned to i and let $\ell := \min_{i \in \mathcal{M}} \ell_i$. Then, for all $i \in \mathcal{M}$ we have $\ell \leq \ell_i \leq \frac{n_i}{n_i - 1} \ell$, where $\frac{n_i}{n_i - 1} = \frac{1}{0} := +\infty$ whenever $n_i = 1$.*

We immediately obtain by Lemma 2.3 the following structure on FLEPT.

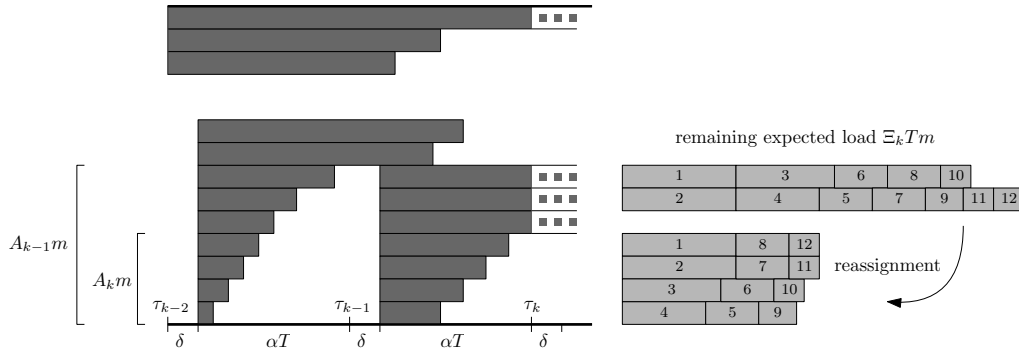
► **Corollary 2.4.** *Given an assignment of jobs to machines induced by FLEPT, we can partition the set of machines into two types of machines: Either there is only a single job assigned to a machine or the expected load of a machine is bounded by 2ℓ . Moreover, ℓ can be bounded from above by the averaged expected load. In particular, if x denotes the total (remaining) expected load and m' is a lower bound on the total number of machines m , then $2\ell \leq 2 \cdot \frac{x}{m'}$.*

Corollary 2.4 will play a central role in showing Theorem 2.1 as FLEPT constitutes an essential part of $\text{LEPT}_{\delta,\alpha}$, which we define now.

► **Definition 2.5** (Policy $\text{LEPT}_{\delta,\alpha}$). *Let $\delta, \alpha > 0$, $k^* := \lfloor \log_2(\frac{2}{3}(\log_2(m)) + 1) \rfloor + 2$ and let $T := 2 \cdot \max\{\frac{1}{m} \sum_{j \in \mathcal{J}} \mathbb{E}[P_j], \max_j \mathbb{E}[P_j]\}$. Moreover, let $\tau_k := k(\delta + \alpha T)$ for $k \in [k^* + 1]$. At the beginning the jobs are assigned according to FLEPT. For $k = 1, \dots, k^* + 1$, $\text{LEPT}_{\delta,\alpha}$ reassigns the jobs that have not started yet before τ_k to the machines that have processed all jobs assigned at previous iterations $0, \dots, k - 1$ by time τ_1, \dots, τ_k , respectively, according to FLEPT. The reassigned jobs may start at time $\tau_k + \delta$ at the earliest.*

We note that in practice it makes sense to use all available machines at each iteration instead of the machines that were available in each previous iteration. Although our policy is limited, we show that in its execution a constant fraction of machines is available in each iteration with high probability. It also simplifies our analysis and matches the bound shown in the next section. Furthermore, observe that $\text{LEPT}_{\delta,\alpha}$ is both a δ -delay policy and a $(\delta + \alpha T)$ -shift policy. Next, let us introduce some quantities which will turn out to be helpful to analyze $\text{LEPT}_{\delta,\alpha}$.

► **Definition 2.6.** Let Ξ_k denote the random variable describing the total expected processing time of the remaining jobs which have not been started at time $< \tau_k$ divided by Tm . Moreover, let A_k denote the random variable describing the fraction of machines which are available at each time τ_1, \dots, τ_k , i.e., the machines have completed all jobs assigned in each iteration $0, \dots, k-1$.



■ **Figure 3** Snippet of $\text{LEPT}_{\delta,\alpha}$: Realizations of jobs observed up to time τ_k are depicted by rectangles in dark grey; the running jobs non-completed by the time of the snippet are indicated by squared dots in dark grey; the expected processing times of the jobs that did not start yet are depicted in light gray. The remaining expected processing time of the twelve light gray jobs is $\Xi_k Tm$. These jobs are reassigned in an FLEPT fashion to the $A_k m$ machines at the bottom at time τ_k , and the first job of each newly formed queue will start at time $\tau_k + \delta$.

A snippet of $\text{LEPT}_{\delta,\alpha}$ together with the introduced notation is illustrated in Figure 3.

Observe that the randomness of Ξ_k occurs only in the set of remaining jobs. We begin with some simple observations.

► **Observation 2.7.** For any k , we have $\Xi_k \leq \Xi_{k-1} \leq 1$ and $A_k \leq A_{k-1} \leq 1$ almost surely.

Next, we want to discuss lower bounds on the expected makespan of an optimal non-anticipatory policy. The first one justifies the use of T in the definition of $\text{LEPT}_{\delta,\alpha}$.

► **Lemma 2.8.** Let $T := 2 \cdot \max\{\frac{1}{m} \sum_{j \in \mathcal{J}} \mathbb{E}[P_j], \max_j \mathbb{E}[P_j]\}$ and ℓ be defined as in Lemma 2.3. Then, we have $2\ell \leq T \leq 2 \cdot \text{OPT}$.

Proof. By Corollary 2.4 we immediately obtain the first inequality as ℓ is a lower bound on the averaged load $\frac{1}{m} \sum_{j \in \mathcal{J}} \mathbb{E}[P_j]$. Clearly, for each realization \mathbf{p} the makespan is bounded from below by $\frac{1}{m} \sum_{j \in \mathcal{J}} p_j$. Hence, taking expectations we obtain $\text{OPT} \geq \frac{1}{m} \sum_{j \in \mathcal{J}} \mathbb{E}[P_j]$. Lastly, in any non-anticipatory policy obviously all jobs must be scheduled non-preemptively. Therefore, $\max_j \mathbb{E}[P_j]$ is another lower bound on OPT . ◀

We obtain another lower bound when only at most m jobs have to be scheduled.

► **Lemma 2.9.** *We have $\mathbb{E}[\max_{j \in \mathcal{J}} P_j] \leq \text{OPT}$.*

Proof. For any realization \mathbf{p} , a lower bound on the optimal makespan for \mathbf{p} is $\max_{j \in \mathcal{J}} p_j$. Taking expectations yields the statement. ◀

We have now set all necessary definitions and lower bounds on the cost of an optimal non-anticipatory policy to devote the remaining part of this section to prove Theorem 2.1. We first derive an upper bound on $\text{LEPT}_{\delta, \alpha}$ in terms of Ξ_{k^*+1} .

► **Lemma 2.10.** *We have that $\text{LEPT}_{\delta, \alpha} \leq \tau_{k^*+1} + \delta + \text{OPT} + \mathbb{E}[\Xi_{k^*+1}] \cdot Tm$.*

Proof. Let $C(\mathbf{p})$ denote the first point in time in realization \mathbf{p} in which all jobs that started before τ_{k^*+1} are completed. We consider an auxiliary policy Π which is identical to $\text{LEPT}_{\delta, \alpha}$ up to time τ_{k^*+1} and starts processing the remaining jobs at time $\max\{\tau_{k^*+1} + \delta, C(\mathbf{p})\}$ on an arbitrary single machine. Clearly, $\text{LEPT}_{\delta, \alpha} \leq \Pi$ since $\text{LEPT}_{\delta, \alpha}$ starts the remaining jobs at time $\tau_{k^*+1} + \delta$, hence, not later than Π and uses at least as many machines as Π . For any realization \mathbf{p} and the starting time of the remaining jobs $S(\mathbf{p})$ we have

$$S(\mathbf{p}) = \tau_{k^*+1} + \max\{\delta, C(\mathbf{p}) - \tau_{k^*+1}\} \leq \tau_{k^*+1} + \max\{\delta, \max_{j \in \mathcal{J}} p_j\} \leq \tau_{k^*+1} + \delta + \max_{j \in \mathcal{J}} p_j.$$

Hence, by Lemma 2.9 the expected starting time is at most $\tau_{k^*+1} + \delta + \text{OPT}$. By definition of Ξ_{k^*+1} the expected remaining load is exactly $\mathbb{E}[\Xi_{k^*+1}] \cdot Tm$. ◀

Due to the derived upper bound it only remains to bound $\mathbb{E}[\Xi_{k^*+1}]$. The next central lemma provides an upper bound on the probability that this quantity is large.

► **Lemma 2.11.** *There exists $\alpha > 1$ such that we have $\mathbb{P}(\Xi_{k^*+1} > \frac{1}{m}) = o(\frac{1}{m})$.*

Let us assume for a moment that Lemma 2.11 is true. We then can prove the main theorem.

Proof of Theorem 2.1. By Lemmas 2.10 and 2.11 and by the law of total expectation we obtain

$$\begin{aligned} \text{LEPT}_{\delta, \alpha} &\leq \tau_{k^*+1} + \delta + \text{OPT} + \mathbb{E}[\Xi_{k^*+1}] \cdot Tm \\ &= \tau_{k^*+1} + \delta + \text{OPT} + \underbrace{\mathbb{P}\left(\Xi_{k^*+1} \leq \frac{1}{m}\right)}_{\leq 1} \cdot \underbrace{\mathbb{E}\left[\Xi_{k^*+1} \mid \Xi_{k^*+1} \leq \frac{1}{m}\right]}_{\leq T} \cdot Tm \\ &\quad + \underbrace{\mathbb{E}\left[\Xi_{k^*+1} \mid \Xi_{k^*+1} > \frac{1}{m}\right]}_{\leq 1} \cdot \underbrace{\mathbb{P}\left(\Xi_{k^*+1} > \frac{1}{m}\right)}_{=o(1)} \cdot m \cdot T \\ &\leq (\alpha T + \delta) \cdot \mathcal{O}(\log \log(m)) + \delta + \text{OPT} + T + o(1) \cdot T \\ &= \mathcal{O}(\log \log(m)) \cdot \text{OPT}, \end{aligned}$$

where the last step follows by Lemma 2.8 and the choice of δ and α . ◀

Let us return to the proof of Lemma 2.11. The high level idea is to use induction to show that in each iteration there is a constant fraction of available machines with high probability and hence, the remaining expected load after k^* iterations is small with high probability. The first lemma provides a stochastic dominance relation of Ξ_k and A_k to binomially distributed random variables in order to simplify calculations.

79:8 Restricted Adaptivity in Stochastic Scheduling

► **Lemma 2.12.** For all $u, \xi \in (0, 1)$, for all $a \in [\frac{1}{2}, 1]$ and for any iteration k we have,

$$\mathbb{P}(\Xi_{k+1} \leq u | \Xi_k \leq \xi, A_k \geq a) \geq \mathbb{P}\left(\frac{2\xi}{am}Y \leq u\right), \text{ where } Y \sim \text{Bin}\left(m, \frac{2\xi}{a\alpha}\right) \quad (1)$$

and

$$\mathbb{P}(A_{k+1} \geq u | \Xi_{k-1} \leq \xi, A_k \geq a) \geq \mathbb{P}\left(\frac{1}{m}Z \geq u\right), \text{ where } Z \sim \text{Bin}\left(\lceil am \rceil, 1 - \frac{2\xi}{a\alpha}\right). \quad (2)$$

Proof sketch. For (2) the key idea is that in each iteration k a machine is available with probability at least $\left(1 - \frac{2\xi}{a\alpha}\right)$ using Corollary 2.4 and Markov's inequality. To show (1) we additionally bound the (normalized) remaining expected load by $\frac{2\xi}{am}$ using Corollary 2.4. ◀

Using Lemma 2.12 we inductively prove probability bounds on Ξ_k and A_k without conditioning on the random variables of the previous iterations. The next lemma handles the base case of the induction stated in Lemma 2.14.

► **Lemma 2.13** (Base case of induction). Let $\gamma_1 = 1$ and $\beta_2 = \frac{3}{4}$. Then, there exists $\epsilon = e^{-\Theta(m^{\frac{1}{3}})}$ such that

$$\mathbb{P}(\Xi_1 \leq \gamma_1) \geq 1 - \epsilon, \quad (3)$$

$$\mathbb{P}(A_2 \geq \beta_2) \geq 1 - 3\epsilon. \quad (4)$$

Proof sketch. The first statement (3) is clear, as $\Xi_1 \leq 1$ almost surely. For the second statement, we first show that A_1 is large with high probability using Corollary 2.4 and the Chernoff bound. This bound can be used together with Lemma 2.12 and the Chernoff bound to show (4). ◀

We use the above statement as the base case of an induction to show the next lemma.

► **Lemma 2.14.** Let $\gamma_1 = 1$, $\gamma_{k+1} = \frac{1}{2}\gamma_k^2$ ($\forall k \geq 1$), $\beta_k = \frac{3}{4} - \frac{2}{\alpha} \sum_{h=1}^{k-2} \gamma_h$ ($\forall k \geq 2$) and let $k^* := \lfloor \log_2\left(\frac{2}{3}(\log_2(m)) + 1\right) \rfloor + 2$. Then, there exists $\psi = \Theta\left(\frac{1}{\log \log(m)}\right)$ and $\epsilon = e^{-\Theta(m^{\frac{1}{3}})}$ such that

$$\mathbb{P}(\Xi_k \leq \gamma_k) \geq 1 - (2^k - 1)\epsilon, \quad \forall k = 1, \dots, k^* \quad (5)$$

$$\mathbb{P}(A_k \geq \beta_k - (k - 2)\psi) \geq 1 - (2^k - 1)\epsilon, \quad \forall k = 2, \dots, k^*. \quad (6)$$

Proof sketch. The doubly exponential decrease of γ_k and the choice of α implies that $\beta_k > \beta_\infty > \frac{5}{8}$. Additionally, the choice of ψ yields $\beta_\infty - (k - 2)\psi \geq \frac{1}{2}$. Thus, we can assume that at each iteration with high probability half of the machines are available. For the induction step we make use of Lemma 2.12, the Chernoff bound and the union bound. ◀

By Lemmas 2.12–2.14 we can now show the probability bound on the remaining expected load at iteration $(k^* + 1)$.

Proof of Lemma 2.11. Let $u = \frac{1}{m}$, $\xi = m^{-\frac{2}{3}}$ and $a = \frac{1}{2}$. Lemma 2.12 (1) implies

$$\mathbb{P}(\Xi_{k^*+1} \leq u | \Xi_{k^*} \leq \xi, A_{k^*} \geq a) \geq \mathbb{P}\left(Y \leq \frac{1}{4}m^{\frac{2}{3}}\right),$$

where $Y \sim \text{Bin}\left(m, \frac{4}{\alpha}m^{-\frac{2}{3}}\right)$. As $\mathbb{E}[Y] = \frac{4}{\alpha}m^{\frac{1}{3}}$ we obtain by applying the Chernoff bound for $\zeta = \frac{\alpha}{16}m^{\frac{1}{3}} - 1 > 0$

$$\mathbb{P}\left(Y \leq \frac{1}{4}m^{\frac{2}{3}}\right) = \mathbb{P}(Y \leq (1 + \zeta) \cdot \mathbb{E}[Y]) \geq \exp\left(-\frac{\mathbb{E}[Y] \cdot \zeta^2}{2 + \zeta}\right) = 1 - \exp(-\Theta(m^{\frac{2}{3}})) \geq 1 - \epsilon,$$

for m large enough. This yields

$$\begin{aligned} \mathbb{P}(\Xi_{k^*+1} \leq u) &\geq \mathbb{P}(\Xi_{k^*+1} \leq u \mid \Xi_{k^*} \leq \xi, A_{k^*} \geq a) \cdot \mathbb{P}(\Xi_{k^*} \leq \xi, A_{k^*} \geq a) \\ &\geq (1 - \epsilon) \cdot (1 - (2^{k^*} - 1)\epsilon - (2^{k^*} - 1)\epsilon) \\ &\geq 1 - (1 + 2 \cdot (2^{k^*} - 1))\epsilon \\ &= 1 - (2^{k^*+1} - 1)\epsilon, \end{aligned}$$

where we used the law of total probability in the first inequality and for the second step we used the union bound and Lemma 2.14. Therefore, as $2^{k^*+1} = \Theta(\log(m))$, we have $\mathbb{P}(\Xi_{k^*+1} > \frac{1}{m}) \cdot m \rightarrow 0$ as $m \rightarrow \infty$. \blacktriangleleft

3 Lower Bound

Throughout this section, we consider an instance I_N with $n = Nm$ jobs over m machines. Each job has processing time $P_j \sim \text{Bernoulli}(\frac{1}{N})$, i.e. $P_j = 1$ with probability $\frac{1}{N}$, and $P_j = 0$ otherwise. The main result of this section is a $\Omega(\delta \log \log(m))$ lower bound on the performance of any δ -delay policy for large values of N . This matches the upper bound obtained in the previous section. Note that the hidden constant in the Ω notation does not depend on the value of $\delta > 0$. For $\delta = \Theta(\text{OPT})$, this implies that no δ -delay policy can improve on the $\log \log m$ performance guarantee of $\text{LEPT}_{\delta, \alpha}$ by more than some constant factor. At the end of the section we show that an analogous result holds for τ -shift policies as well.

► **Theorem 3.1.** *Let $\delta \leq 1$. For instance I_N let $\text{OPT}_{\delta}^{\text{DELAY}}$ and OPT denote the value of an optimal δ -delay policy and of an optimal non-anticipatory policy, respectively. Then, for $N = \Omega(\sqrt{m})$ we have*

$$\frac{\text{OPT}_{\delta}^{\text{DELAY}}}{\text{OPT}} = \Omega(\delta \cdot \log \log(m)).$$

The proof is split into two main lemmas. The first one relates the expected makespan of an optimal δ -delay policy to the expected makespan of an optimal 1-delay policy.

► **Lemma 3.2.** *Assume $\frac{1}{\delta} \in \mathbb{N}$. Then, we have $\text{OPT}_{\delta}^{\text{DELAY}} \geq \delta \cdot \text{OPT}_1$.*

The second lemma shows that OPT_1 grows doubly logarithmically with m .

► **Lemma 3.3.** *For $N = \Omega(\sqrt{m})$ it holds $\text{OPT}_1 = \Omega(\log \log(m))$.*

Let us assume for now that the above lemmas hold. Then, we simply need to show that $\text{OPT} = O(1)$ to prove the theorem.

Proof of Theorem 3.1. On the one hand, Lemma 3.2 and Lemma 3.3 imply $\text{OPT}_{\delta}^{\text{DELAY}} = \Omega(\delta \cdot \log \log(m))$. On the other hand, we can use the List Scheduling policy (*LS*) due to Graham [11] to obtain an upper bound on the value of an optimal non-anticipatory policy.

Whenever a machine becomes idle, LS schedules any non-scheduled job on it. For any fixed realization $\mathbf{p} = (p_j)_{j \in [Nm]}$ we obtain for its makespan $C_{\max}^{LS}(\mathbf{p}) = \left\lceil \frac{1}{m} \sum_{j \in [Nm]} p_j \right\rceil \leq 1 + \frac{1}{m} \sum_{j \in [Nm]} p_j$. As a result, taking expectations on both sides yields

$$\text{OPT} \leq \mathbb{E}[C_{\max}^{LS}] \leq 1 + \frac{1}{m} \sum_{j \in [Nm]} \mathbb{E}[P_j] = 1 + \frac{1}{m} \cdot Nm \cdot \frac{1}{N} = 2,$$

concluding the proof of the theorem. \blacktriangleleft

To prove the lemmas, we first make an observation on the structure of optimal δ -delay policies. When we execute a set of Bernoulli jobs on a machine, we immediately observe whether one of the jobs was a long job (i.e., $p_j = 1$), and also the number of vanishing jobs (i.e., $p_j = 0$) that have already been executed. This indicates that optimal δ -delay policies do not insert deliberate idle time in the schedule (since waiting does not provide any information on running jobs), and for the case $\frac{1}{\delta} \in \mathbb{N}$, they may only take reassignment decisions at times of the form $k\delta$ for $k \in \mathbb{N}$. We call policies with this property *δ -active*.

Proof of Lemma 3.2. The starting time of each job in $\text{OPT}_{\delta}^{\text{DELAY}}$ is an integer multiple of δ , because $1/\delta \in \mathbb{N}$ and $\text{OPT}_{\delta}^{\text{DELAY}}$ is δ -active. Let $J_{ki}(\mathbf{p})$ denote the set of jobs started on machine i at time $k\delta$ by $\text{OPT}_{\delta}^{\text{DELAY}}$, for a realization $\mathbf{p} \in \{0, 1\}^n$ of the processing times. $J_{ki}(\mathbf{p})$ may contain many vanishing jobs executed at time $t = k\delta$, and at most one long job executed during the time interval $[k\delta, k\delta + 1)$. It is easy to construct a 1-delay policy (call it Π_1) that executes the same set of jobs $J_{ki}(\mathbf{p})$ during the interval $[k, k + 1)$ on machine i , by taking at time $k - 1$ the same reassignment decisions as $\text{OPT}_{\delta}^{\text{DELAY}}$ takes at time $(k - 1)\delta$, and by waiting until time $t = k$ to execute the reassigned jobs. In both schedules, the makespan is caused by the same long job (if there is at least one long job). Its starting time is $\text{OPT}_{\delta}^{\text{DELAY}}(\mathbf{p}) - 1$ in the optimal δ -delay policy and $\frac{1}{\delta}(\text{OPT}_{\delta}^{\text{DELAY}}(\mathbf{p}) - 1)$ in the policy Π_1 . Hence the policy Π_1 has makespan $\Pi_1(\mathbf{p}) = \frac{1}{\delta} \cdot (\text{OPT}_{\delta}^{\text{DELAY}}(\mathbf{p}) - 1) + 1$ for any realization $\mathbf{p} \neq \mathbf{0}$, and $\Pi_1(\mathbf{p}) = \text{OPT}_{\delta}^{\text{DELAY}}(\mathbf{p}) = 0$ if $\mathbf{p} = \mathbf{0}$. Taking expectations yields

$$\text{OPT}_1 \leq \mathbb{E}[\Pi_1(\mathbf{p})] = \frac{1}{\delta} \cdot \text{OPT}_{\delta}^{\text{DELAY}} + \mathbb{P}(\mathbf{p} \neq \mathbf{0}) \cdot \left(1 - \frac{1}{\delta}\right) \leq \frac{1}{\delta} \cdot \text{OPT}_{\delta}^{\text{DELAY}},$$

where we have used the fact that $\delta \leq 1$. This implies $\text{OPT}_{\delta}^{\text{DELAY}} \geq \delta \cdot \text{OPT}_1$. \blacktriangleleft

This lemma allows us to work with 1-delay policies, which are easier to handle: At all times $t \in \mathbb{N}$, a 1-active policy observes the set of jobs non-started yet at time $t - 1 + \epsilon$ (for an infinitesimal small $\epsilon > 0$) and reassigns them to *any* machine, on which they will start at time t at the earliest: we call it an *iteration*.

We denote by R_t the random variable describing the number of remaining jobs at time $t \in \mathbb{N}_0$, before OPT_1 runs the jobs, and by $\Lambda_t = \frac{R_t}{Nm}$ the fraction of remaining jobs at time t . For the initial state we have $\Lambda_0 = 1$ (a.s.). Not surprisingly, the optimal policy balances the remaining jobs as evenly as possible on the m machines.

► **Proposition 3.4.** *In iteration t , OPT_1 assigns the remaining $\Lambda_t Nm$ jobs by balancing the load as evenly as possible, i.e., each machine receives $\lceil \Lambda_t N \rceil$ or $\lfloor \Lambda_t N \rfloor$ jobs.*

Proof sketch. Consider a realization of the jobs started before time $t - 1$ for some $t \in \mathbb{N}$, and in which r jobs remain at time $t - 1$. A 1-active policy must reassign the r jobs to the m machines. By moving jobs between two machines, one can show that the balancing policy, which assigns $\lfloor \frac{r}{m} \rfloor$ or $\lceil \frac{r}{m} \rceil$ jobs to each machine, minimizes the (random) number

of remaining jobs at time t for the order of stochastic dominance, in the class of 1-active policies. Then, the optimality of the balancing policy follows from the fact that the expected cost-to-go from iteration t , $r \mapsto \mathbb{E}[\text{OPT}_1 - t | R_t = r]$ is monotone decreasing with respect to the number of remaining jobs. ◀

For notational convenience let $\lfloor \Lambda_t N \rfloor_i$ denote the number of jobs assigned to machine i by OPT_1 . By independence of the processing times, the number of jobs that must be drawn before picking a long job is geometrically distributed with parameter $\frac{1}{N}$. Consequently, we obtain the following observation.

► **Observation 3.5.** *For $i \in [m]$ let $G_i \sim \text{Geom}(\frac{1}{N})$ be i.i.d. random variables. Then, we have*

$$\Lambda_{t+1} \stackrel{d}{=} \frac{1}{Nm} \sum_{i=1}^m (\lfloor \Lambda_t N \rfloor_i - G_i)_+.$$

We can now prove that OPT_1 is of order $\Omega(\log \log(m))$. To do this, we first need a lemma showing that Λ_t converges quadratically to 0.

► **Lemma 3.6.** *For $N = \Omega(\sqrt{m})$ and $t \in \{1, \dots, \lfloor \log_2(\frac{1}{4} \log_{2e}(m)) \rfloor\}$ we have*

$$\mathbb{P}(\Lambda_t \geq (2e)^{1-2^t}) \geq (1 - e^{-2\sqrt{m}})^t.$$

A rigorous proof of this lemma is proved in the appendix of the full version [30]. For now, we just explain the intuition behind the quadratic convergence of Λ_t to 0 in expectation, by taking a (hand-wavy look) at the conditional expectation $\mathbb{E}[\Lambda_{t+1} | \Lambda_t = \lambda]$ for large values of N . Using that $\frac{\lfloor \lambda N \rfloor}{N} \rightarrow \lambda$ and the well-known fact that $\frac{G_i}{N}$ converges in distribution to an exponential random variable $X \sim \text{Exp}(1)$, we see that when $N \rightarrow \infty$, $\mathbb{E}[\Lambda_{t+1} | \Lambda_t = \lambda]$ should approach $\mathbb{E}[(\lambda - X)_+] = \int_{x=0}^{\lambda} (\lambda - x)e^{-x} dx = \lambda + e^{-\lambda} - 1$. Then, the quadratic convergence of Λ_t is suggested by the inequalities $\frac{\lambda^2}{e} \leq \lambda + e^{-\lambda} - 1 \leq \frac{\lambda^2}{2}$, which hold for all $\lambda \in [0, 1]$.

With this lemma, we obtain a short proof for Lemma 3.3.

Proof of Lemma 3.3. By Lemma 3.6 we obtain for $t = \Omega(\log \log(m))$ and $N = \Omega(\sqrt{m})$

$$\text{OPT}_1 \geq \mathbb{P}(C_{\max} \geq t) \cdot t \geq \mathbb{P}(\Lambda_t \geq (2e)^{1-2^t}) \cdot t \geq \underbrace{\left(1 - e^{-2\sqrt{m}}\right)^t}_{\xrightarrow{m \rightarrow \infty} 1} \cdot t = \Omega(\log \log(m)). \quad \blacktriangleleft$$

A similar result can be shown for τ -shift policies, for the same instance I_N .

► **Theorem 3.7.** *Let $\tau \leq 1$, such that $\frac{1}{\tau} \in \mathbb{N}$. For instance I_N let $\text{OPT}_\tau^{\text{SHIFT}}$ and OPT denote the value of an optimal τ -shift policy and of an optimal non-anticipatory policy, respectively. Then, for $N = \Omega(\sqrt{m})$ we have*

$$\frac{\text{OPT}_\tau^{\text{SHIFT}}}{\text{OPT}} = \Omega(\tau \cdot \log \log(m)).$$

Proof. Similarly as for the case of δ -delay policies, for $1/\tau \in \mathbb{N}$ it is clear that an optimal τ -shift policy for instance I_N must be τ -active. Therefore, the optimal τ -active policy coincides with both the optimal τ -shift and the optimal τ -delay policy. This shows that $\text{OPT}_\tau^{\text{SHIFT}} = \text{OPT}_\tau^{\text{DELAY}}$, and the result follows from Theorem 3.1. ◀

4 Conclusion

We considered the stochastic optimization problem of minimizing the expected makespan on parallel identical machines. While any list scheduling policy is a constant factor approximation, the performance guarantee of all fixed assignment policies is at least $\Omega\left(\frac{\log m}{\log \log m}\right)$. We introduced two classes of policies to establish a happy medium between the two extremes of adaptive and non-adaptive policies. The policy $\text{LEPT}_{\delta, \alpha}$, which is both a δ -delay and a τ -shift policy, was shown to have performance guarantee of $\mathcal{O}(\log \log m)$ if δ and τ are in the scale of the instance. Moreover, we provided a matching lower bound for $\delta, \tau = \Theta(\text{OPT})$. Therefore, $\text{LEPT}_{\delta, \alpha}$ improves upon the performance of an optimal fixed assignment policy using a small amount of adaptivity. Moreover, there exists no δ -delay or τ -shift policy beating its performance guarantee by more than a constant.

For the case of $\delta, \tau = \mathcal{O}\left(\frac{1}{\log \log m}\right)$, Theorem 3.1 gives a constant lower bound, while Theorem 2.1 only gives a doubly logarithmic upper bound. An open question is whether a constant approximation guarantee is possible in this case.

A possible future line of research is the analysis of δ -delay and τ -shift policies for stochastic scheduling problems with other numerous objectives, different machine environments as well as various job characteristics. Moreover, it would be interesting to design other non-anticipatory policies whose adaptivity can be controlled.

References

- 1 Gagan Aggarwal, Rajeev Motwani, and An Zhu. The load rebalancing problem. *Journal of Algorithms*, 60(1):42–59, 2006.
- 2 Susanne Albers and Matthias Hellwig. On the value of job migration in online makespan minimization. *Algorithmica*, 79(2):598–623, 2017.
- 3 Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
- 4 B.P. Berg and B.T. Denton. Fast approximation methods for online scheduling of outpatient procedure centers. *INFORMS Journal on Computing*, 29(4):631–644, 2017.
- 5 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 657–668, 2013.
- 6 Anindya De, Sanjeev Khanna, Huan Li, and Hesam Nikpey. An efficient PTAS for stochastic load balancing with poisson jobs. In *47th International Colloquium on Automata, Languages, and Programming*, volume 168 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 7 Brian T. Denton, Andrew J. Miller, Hari J. Balasubramanian, and Todd R. Huschka. Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations Research*, 58(4-1):802–816, 2010.
- 8 Matthias Englert, Deniz Ozmen, and Matthias Westermann. The power of reordering for online minimum makespan scheduling. *SIAM Journal on Computing*, 43(3):1220–1237, 2014.
- 9 Michael R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness, 1979.
- 10 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science*, pages 579–586. IEEE, 1999.
- 11 Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.
- 12 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

- 13 Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979.
- 14 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. *Mathematics of Operations Research*, 46(1):115–133, 2021.
- 15 Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Greed works—online algorithms for unrelated machine stochastic scheduling. *Mathematics of Operations Research*, 45(2):497–516, 2020.
- 16 Willy Herroelen and Roel Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.
- 17 Dorit S. Hochbaum. Various notions of approximations: Good, better, best and more. *Approximation algorithms for NP-hard problems*, 1997.
- 18 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
- 19 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.
- 20 David Isern, David Sánchez, and Antonio Moreno. Agents applied in health care: A review. *International journal of medical informatics*, 79(3):145–166, 2010.
- 21 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.
- 22 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020.
- 23 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.
- 24 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1):259–271, 1990.
- 25 Nicole Megow, Marc Uetz, and Tjark Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.
- 26 Rolf H. Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems I—general strategies. *Zeitschrift für Operations Research*, 28(7):193–260, 1984.
- 27 Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *Journal of the ACM*, 46(6):924–942, 1999.
- 28 Marco Molinaro. Stochastic lp load balancing and moment problems via the l-function method. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 343–354. SIAM, 2019.
- 29 Guillaume Sagnol, Christoph Barner, Ralf Borndörfer, Mickaël Grima, Mathees Seeling, Claudia Spies, and Klaus Wernecke. Robust allocation of operating rooms: A cutting plane approach to handle lognormal case durations. *European Journal of Operational Research*, 271(2):420–435, 2018.
- 30 Guillaume Sagnol and Daniel Schmidt genannt Waldschmidt. Restricted adaptivity in stochastic scheduling, 2021. [arXiv:2106.15393](https://arxiv.org/abs/2106.15393).
- 31 Guillaume Sagnol, Daniel Schmidt genannt Waldschmidt, and Alexander Tesch. The price of fixed assignments in stochastic extensible bin packing. In *International Workshop on Approximation and Online Algorithms*, pages 327–347. Springer, 2018.
- 32 Sartaj K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.
- 33 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

79:14 Restricted Adaptivity in Stochastic Scheduling

- 34 Andreas S. Schulz. Stochastic online scheduling revisited. In *International Conference on Combinatorial Optimization and Applications*, pages 448–457. Springer, 2008.
- 35 Martin Skutella, Maxim Sviridenko, and Marc Uetz. Unrelated machine scheduling with stochastic processing times. *Mathematics of Operations Research*, 41(3):851–864, 2016.
- 36 Guanlian Xiao, Willem van Jaarsveld, Ming Dong, and Joris van de Klundert. Models, algorithms and performance analysis for adaptive operating room scheduling. *International Journal of Production Research*, 56(4):1389–1413, 2018.

A General Framework for Enumerating Equivalence Classes of Solutions

Yishu Wang ✉

Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France
Inria Grenoble Rhône-Alpes, Villeurbanne, France

Arnaud Mary ✉

Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France
Inria Grenoble Rhône-Alpes, Villeurbanne, France

Marie-France Sagot ✉

Inria Grenoble Rhône-Alpes, Villeurbanne, France
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France

Blerina Sinaimeri ✉

Luiss University, Rome, Italy
ERABLE team, Inria Grenoble Rhône-Alpes, Villeurbanne, France

Abstract

When a problem has more than one solution, it is often important, depending on the underlying context, to enumerate (i.e., to list) them all. Even when the enumeration can be done in polynomial delay, that is, spending no more than polynomial time to go from one solution to the next, this can be costly as the number of solutions themselves may be huge, including sometimes exponential. Furthermore, depending on the application, many of these solutions can be considered equivalent. The problem of an efficient enumeration of the equivalence classes or of one representative per class (without generating all the solutions), although identified as a need in many areas, has been addressed only for very few specific cases. In this paper, we provide a general framework that solves this problem in polynomial delay for a wide variety of contexts, including optimization ones that can be addressed by dynamic programming algorithms, and for certain types of equivalence relations between solutions.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Mathematics of computing → Graph enumeration; Theory of computation → Backtracking

Keywords and phrases Enumeration algorithms, Equivalence relation, Dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.80

Related Version *Full Version*: <https://arxiv.org/abs/2004.12143>

1 Introduction

Enumerating the solutions of an optimization problem solved by a dynamic programming algorithm (DP-algorithm) is a classical and well-known question. However, many enumeration problems have a huge number of solutions in practice, which might be an issue. From a computational point of view, since the number of solutions is a lower bound on the time complexity of any enumeration algorithm, it might make the algorithm impractical on real instances. Furthermore, even if the number of solutions is reasonable enough to be enumerated, the purpose of some applications is to give the output of the algorithm to a human specialist (this is necessary, for example, when some of the constraints of the problem are subjective and cannot be modeled).



© Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinaimeri;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 80; pp. 80:1–80:14

Leibniz International Proceedings in Informatics



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

Indeed, one of the advantages of an enumeration algorithm compared to an optimization one which in general outputs only one optimal solution, is to be able to understand the space of solutions. While this is important in many cases, no human can understand an output composed of billions of solutions.

The approach generally used to address this consists of enumerating all solutions, and then applying some type of clustering (grouping) algorithm to the set of optimal solutions. The final output presented to the user would then be some “representative description” of the clusters (groups) themselves. However, since the number of solutions is a lower bound for the total execution time of any enumeration algorithm, the first step of such a strategy becomes impossible when the number of solutions is too big. A natural question is then whether it would be possible to enumerate directly what we just called a “representative description” of the clusters of solutions. This could be for instance an element per cluster. Sometimes a cluster can also be seen as a set of characteristics that the solutions within the cluster share. In such a case, the representative description of a cluster could then be such a set of characteristics. A particularly convenient situation is however when the clusters correspond to equivalence classes of an equivalence relation over the set of solutions that we could establish *a priori*. The output could be in this case the quotient space of the equivalence relation. Notice that the enumeration of equivalence classes of solutions is a combinatorial problem that could be solved exactly given a well-defined equivalence relation, and unlike data analysis methods such as incremental clustering, it does not require the definition of a similarity or dissimilarity measure between solutions which, depending on the mathematical nature of the solutions (numerical values, graphs, functions on graphs, etc.), can be difficult to define or costly to compute.

The problem this paper addresses is how to perform the task of enumerating equivalence classes of solutions with polynomial delay for a wide variety of problems (including optimization problems solved by dynamic programming algorithms), for certain types of equivalence relations between solutions.

The problem of enumerating equivalence classes, and particularly the generation of representative solutions is a challenge in the context of enumeration algorithms. It has been identified as a need in different areas, such as Genome Rearrangements [10], Artificial Intelligence [1] or Pattern Matching [7, 26]. It was listed as an important open problem in a recent Dagstuhl workshop on “Algorithmic Enumeration: Output-sensitive, Input-Sensitive, Parameterized, Approximative” (see, e.g., Sections 4.2 and 4.10 in [15]). To the best of our knowledge, this challenge has been addressed only for some few specific problems in the literature (e.g., [2, 10, 24, 25]).

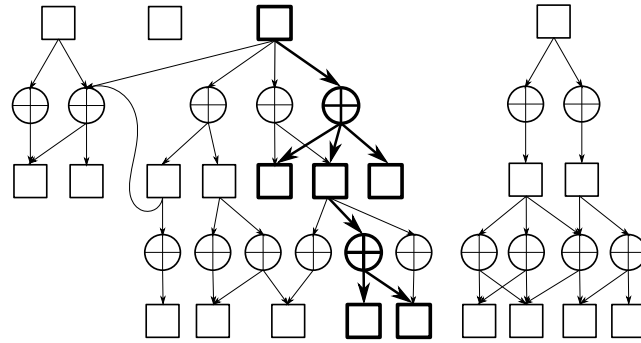
To enumerate equivalence classes, we go through an intermediate problem, namely the enumeration of colored subtrees in acyclic decomposable AND/OR graphs (ad-AND/OR graph). The paper is organized as follows: Section 2 provides an algorithm to enumerate with polynomial delay colored subtrees in ad-AND/OR graphs; Section 3 details how this algorithm applies to the enumeration of equivalence classes in DP-problems. In that direction, we present some examples from well-known optimization problems in the literature. Finally, in Section 4 we conclude with some open problems.

2 Enumeration of colored subtrees in an ad-AND/OR graph

2.1 AND/OR graphs and solution subtrees

An *AND/OR graph* (see, for example [23, 27]) is a well-known structure in the field of Logic and Artificial Intelligence (AI) that represents problem solving and problem decomposition. In this paper, we consider a particular flavor of AND/OR graphs known as *explicit AND/OR graphs for trees* [12].

This is a directed acyclic graph (DAG) G which explicitly represents an *AND/OR state space* for solving a certain problem by decomposing it into subproblems. The set of nodes (or states) $S := V(G)$ contains OR and AND nodes (the OR nodes represent alternative ways for solving the problem while the AND nodes represent problem decomposition into subproblems, all of which need to be solved). There is a set of goal nodes $S_g \subseteq S$ and a set of start nodes $S_0 \subseteq S$ representing respectively the terminal states and the initial states. The children (out-neighbors) of an OR node are AND nodes, and the children of an AND node are OR nodes or goal nodes. We say that a node is an OR^+ node when it is either an OR node or a goal node. Furthermore, the AND/OR graphs that we consider must have the property of being *decomposable* (they can model a problem for which every decomposition yields disjoint subproblems that can be solved independently): for any AND node, the sets of nodes that are reachable from each one of its child nodes are pairwise disjoint. The example graph in Figure 1 is decomposable.



■ **Figure 1** An acyclic decomposable AND/OR graph with four start nodes. Squares are OR^+ nodes (OR nodes or goal nodes); crossed circles are AND nodes. One solution subtree of size 8 is shown in bold.

Formally, in this paper, any graph that satisfies the properties in Definition 1 will be called an ad-AND/OR graph. Notice that this definition corresponds only to a particular case of the general AND/OR graphs in the AI literature; the latter may be neither acyclic nor decomposable.

► **Definition 1** (ad-AND/OR graph). *A directed graph G is an acyclic decomposable AND/OR graph, henceforth denoted by ad-AND/OR graph, if it satisfies the following:*

- G is a DAG.
- G is bipartite: its node set $V(G)$ can be partitioned into $(\mathcal{A}, \mathcal{O})$ so that all arcs of G are between these two sets. Nodes in \mathcal{A} are called AND nodes; nodes in \mathcal{O} are called OR^+ nodes.
- Every AND node has in-degree at least one and out-degree at least one. The set of nodes with out-degree zero is then a subset of \mathcal{O} and is called the set of goal nodes; the remaining OR^+ nodes are simply the OR nodes. The subset of OR nodes of in-degree zero is the set of start nodes.
- G is decomposable: for any AND node, the sets of nodes that are reachable from each one of its child nodes are pairwise disjoint.

► **Definition 2** (solution subtree). *A solution subtree T of an ad-AND/OR graph G is a subgraph of G which: (1) contains exactly one start node; (2) for any OR node in T it contains one of its child nodes in G , and for any AND node in T it contains all its children in G .*

It is immediate to see that a solution subtree is indeed a subtree of G : it is a rooted tree, the root of which is a start node. If we would drop the requirement of G being decomposable, the object defined in Definition 2 would not be guaranteed to be a tree. One solution subtree of the example graph in Figure 1 is shown in bold.

The set of all solution subtrees of G is denoted by $\mathbb{T}(G)$. Given an ad-AND/OR graph G , counting the number of its solution subtrees and enumerating all solution subtrees can be solved by folklore approaches based on depth-first search (DFS).

Before going further, we recall that the motivation of this paper is concerned with solutions of dynamic programming problems. The correspondence between the solutions of DP-style recurrence equations and the solution subtrees of general AND/OR graphs has been formally proven in [16]. In the case where the underlying graph is acyclic, the recurrence equations can be solved efficiently by DP-algorithms. While we will now concentrate on the solution subtrees of an ad-AND/OR graph and on the equivalence classes of solution subtrees, we will demonstrate in Section 3 how to apply our algorithms to analyze equivalence classes of solutions of a very general class of problems solvable by DP. It is important to point out that solution subtrees of general AND/OR graphs are equivalent to various other well-known formalisms, e.g., acceptance trees of a nondeterministic tree automata, languages of regular tree grammars, complete subcircuits of tropical circuits. The reader who is more familiar with those may also find this paper interesting even outside of a dynamic programming context.

2.2 Equivalence classes

Let G be an ad-AND/OR graph. Let C be an **ordered** set of colors. We will consider equivalence relations on the set of solution subtrees of G which are based on a local comparison of the colors of the OR^+ nodes. Intuitively, two OR^+ nodes having the same color represent two alternative ways of solving the problem that can be considered equivalent.

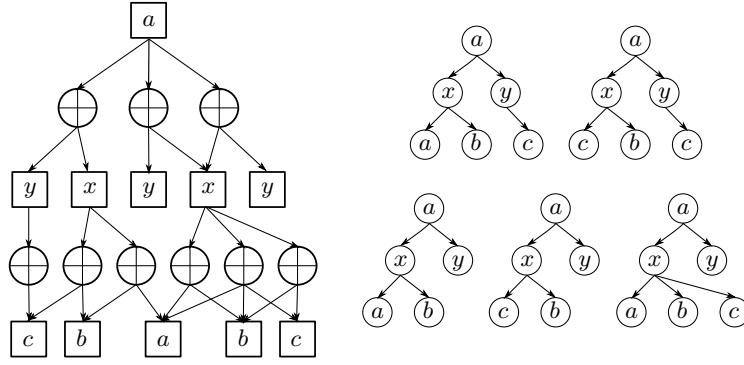
► **Definition 3** (e-coloring). *An ad-AND/OR graph G is e-colored if its OR^+ nodes are colored in such a way that for any AND node all its children have distinct colors.*

Notations. If s is a OR^+ node of G , we denote by $c(s)$ its color. If s is an AND node, we denote by $\tilde{C}(s)$ the tuple of colors of the children of s sorted in increasing order of the colors. If $T_1 \in \mathbb{T}(G)$ is a solution subtree of G , we use the notation $\pi(T_1)$ for the result of contracting the AND nodes in T_1 : for each OR node s of T , contract the only child node of s in T (i.e., remove the child and connect s to each one of its “grandchildren”).

► **Definition 4** (equivalence class). *A node-colored rooted tree T is an equivalence class of solution subtrees of an e-colored ad-AND/OR graph G (or shortly an equivalence class of G) if there exists a solution subtree T_1 of G such that $\pi(T_1)$ is equal to T . Such a T_1 is said to be a solution subtree belonging to the class T .*

More notations. We denote by $\mathbb{C}(G)$ the set of equivalence classes of G . The notation π can be seen as a function $\pi: \mathbb{T}(G) \rightarrow \mathbb{C}(G)$. We denote by $\pi^{-1}(T) := \{T_1 \in \mathbb{T}(G) \mid \pi(T_1) = T\}$ the subset of solution subtrees of G belonging to the class T . The notations $c(s)$ and $\tilde{C}(s)$ are naturally extended to the case where s is a node in an equivalence class T . The root node of a rooted tree T is denoted by $r(T)$. The set of the children of a node s is denoted by $Ch(s)$.

An example of an e-colored ad-AND/OR graph with five equivalence classes is given in Figure 2.



■ **Figure 2** An e-colored ad-AND/OR graph and its five equivalence classes. The colors of the OR^+ nodes are written inside the squares.

2.3 Enumerating equivalence classes

Given an e-colored ad-AND/OR graph G , we propose a polynomial delay algorithm to enumerate all equivalence classes of G . Given a total ordering c_1, \dots, c_m of the colors of G , we define a total ordering \prec over $\mathbb{C}(G)$, the set of equivalence classes of G . If T and T' have their roots colored differently, we say that T is smaller than T' , denoted by $T \prec T'$, if the root color of T precedes the one of T' . If T and T' have the same root color, let (T_1, \dots, T_k) (resp. (T'_1, \dots, T'_k)) be the child subtrees of $r(T)$ (resp. $r(T')$) sorted recursively with respect to \prec . We then say that T is smaller than T' if the tuple (T_1, \dots, T_k) is lexicographically smaller than (T'_1, \dots, T'_k) , i.e., if $T_i \prec T'_i$ with i being the smallest index such that $T_i \neq T'_i$. We also assume that \emptyset is smaller than any tree, and therefore a single node tree colored with color c comes before any other tree whose root is colored with c in \prec .

2.3.1 Definitions and notations

Recall that given an AND-node x , $\tilde{C}(x)$ is the tuple of colors of the children of x sorted in increasing order. Given an OR-node o , we denote by $\mathcal{T}(o)$ the set of color tuples of its children, i.e., $\mathcal{T}(o) := \{\tilde{C}(x) : x \in \text{Ch}(o)\}$. In other words, a color tuple (c_1, \dots, c_j) belongs to $\mathcal{T}(o)$ if o has an AND-child node whose children are colored with (c_1, \dots, c_j) . If we consider an equivalence class T of $\mathbb{C}(G/\{o\})$ rooted at o , the tuples of $\mathcal{T}(o)$ are precisely the possible colorings of the children of $r(T)$. Indeed, if the AND-child node $x \in \text{Ch}(o)$ is chosen in a solution subtree, then $\tilde{C}(x)$ will be the colors of the children of o in that solution. Notice that several AND-children nodes of o may have the same color tuple.

We extend this definition to a set \mathcal{O} of OR^+ nodes with $\mathcal{T}(\mathcal{O}) = \bigcup_{o \in \mathcal{O}} \mathcal{T}(o)$. In the same way, $t = (c_1, \dots, c_j)$ is a color tuple of $\mathcal{T}(\mathcal{O})$ if and only if there exists an equivalence class T of $\mathbb{C}(G/\mathcal{O})$ such that the children of $r(T)$ are colored with (c_1, \dots, c_j) . Given a set \mathcal{O} of OR^+ nodes, we denote by $t_1, \dots, t_{|\mathcal{T}(\mathcal{O})|}$ the different color tuples of $\mathcal{T}(\mathcal{O})$ ordered lexicographically, and we denote by $\text{Ch}^\ell(\mathcal{O})$ the set of AND-nodes in $\text{Ch}(\mathcal{O})$ whose color tuple is t_ℓ , i.e., $\text{Ch}^\ell(\mathcal{O}) = \{x \in \text{Ch}(\mathcal{O}) : \tilde{C}(x) = t_\ell\}$. The sets $\text{Ch}^1(\mathcal{O}), \dots, \text{Ch}^{|\mathcal{T}(\mathcal{O})|}(\mathcal{O})$ form a partition of $\text{Ch}(\mathcal{O})$, each part corresponding to a color tuple $t_i \in \mathcal{T}(\mathcal{O})$.

Finally, given a color tuple $t_\ell := (c_1, \dots, c_j) \in \mathcal{T}(\mathcal{O})$, for each $i \leq j$, by Definition 3, each node of $\text{Ch}^\ell(\mathcal{O})$ has exactly one child with color c_i . We denote by C_i^ℓ the set of children of $\text{Ch}^\ell(\mathcal{O})$ colored with c_i , i.e., $C_i^\ell = \{o \in \text{Ch}(x) : x \in \text{Ch}^\ell(\mathcal{O}), c(o) = c_i\}$ (it is a set of “grandchildren” of \mathcal{O}).

■ **Algorithm 1** Next solution.

```

1 Input: A set  $\mathcal{O}$  of  $\text{OR}^+$  nodes having all the same color  $c$  and an equivalence class  $T$ 
   of  $G/\mathcal{O}$ 
2 Output: The equivalence class  $T'$  of  $G/\mathcal{O}$  that follows  $T$  w.r.t the  $\prec$  ordering.
3 Function  $\text{Next}(T, \mathcal{O})$ :
4   if  $T = \emptyset$  and  $\mathcal{O}$  contains terminal nodes then
5     | Return A tree with a single root node colored with  $c$ 
6   end
7    $r \leftarrow 0$ 
8   if  $T = \emptyset$  or  $T$  is a single node tree then
9     |  $r \leftarrow r + 1$ 
10    | if  $r > |\mathcal{T}(\mathcal{O})|$  then
11      | | Return  $\perp$ 
12    | end
13    | Let  $(c_1, \dots, c_j)$  be the color tuple of  $t_r \in \mathcal{T}(\mathcal{O})$  and let  $T_1, \dots, T_j \leftarrow \emptyset$ 
14    |  $\mathcal{O}_1 \leftarrow C_1^r$ 
15    |  $\ell \leftarrow 1$ 
16  else
17    | Let  $r$  be such that  $t_r \in \mathcal{T}(\mathcal{O})$  is the root color tuple  $\tilde{C}(r(T))$ 
18    | Let  $(T_1, \dots, T_j)$  be the child subtrees of  $r(T)$ , the roots of which are colored
      | respectively with  $(c_1, \dots, c_j) := t_r$ 
19    | For all  $i \leq j$ , let  $\mathcal{O}_i \subseteq C_i^r$  be the set of nodes in  $C_i^r$  compatible with
      |  $(T_1, \dots, T_{i-1})$ 
20    | Let  $\ell$  be the largest index  $i \leq j$  such that  $\text{Next}(T_i, \mathcal{O}_i) \neq \perp$  if such index
      | exists. Otherwise,  $T \leftarrow \emptyset$  and go to line 8
21  end
22   $T_\ell \leftarrow \text{Next}(T_\ell, \mathcal{O}_\ell)$ 
23  for  $\ell < i \leq j$  do
24    | Let  $\mathcal{O}_i \subseteq C_i^r$  be the set of nodes in  $C_i^r$  compatible with  $(T_1, \dots, T_{i-1})$ 
25    |  $T_i \leftarrow \text{Next}(\emptyset, \mathcal{O}_i)$ 
26  end
27  Return A tree with root color  $c$  and root child subtrees  $(T_1, \dots, T_j)$ 

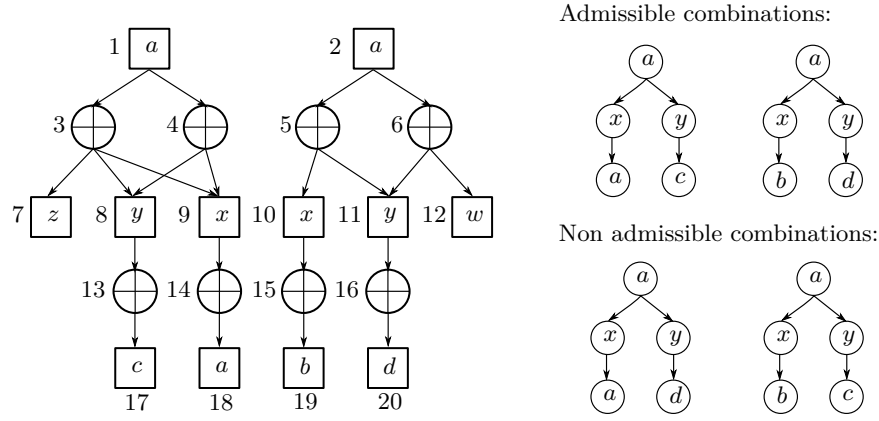
```

In the left panel of Figure 3, an example graph is shown where each node is labeled by an integer. The colors are, in increasing order, w , x , y , and z . For $\mathcal{O} = \{1, 2\}$, the set $\mathcal{T}(\mathcal{O})$ contains the three tuples $t_1 = (w, y)$, $t_2 = (x, y)$, and $t_3 = (x, y, z)$. We have $Ch^1(\mathcal{O}) = \{6\}$, $Ch^2(\mathcal{O}) = \{4, 5\}$, and $Ch^3(\mathcal{O}) = \{3\}$. The sets C_i^ℓ are $C_1^1 = \{12\}$, $C_2^1 = \{11\}$, $C_1^2 = \{9, 10\}$, $C_2^2 = \{8, 11\}$, $C_1^3 = \{9\}$, $C_2^3 = \{8\}$, and $C_3^3 = \{7\}$.

2.3.2 Algorithm description

Notice that by definition of \prec , given a set of OR^+ nodes \mathcal{O} , all having the same color c , and a color tuple $t_\ell \in \mathcal{T}(\mathcal{O})$, all equivalence classes T of G/\mathcal{O} such that $\tilde{C}(r(T)) = t_\ell$ are consecutive with respect to \prec .

The algorithm outputs the equivalence classes in ascending order with respect to \prec . Given an equivalence class T of G/\mathcal{O} for a set of OR^+ nodes \mathcal{O} of color c , it will output the equivalence class T' of G/\mathcal{O} that succeeds T w.r.t. \prec if it exists or output the symbol \perp if T is the last solution.



■ **Figure 3** Left panel: An e-colored ad-AND/OR graph. Right panel: For $\mathcal{O} = \{1, 2\}$, there are four combinations between $\mathbb{C}(G/C_1^2)$ and $\mathbb{C}(G/C_2^2)$; only two of them are admissible.

Assume that the children of $r(T)$ are colored with the *root color tuple* $(c_1, \dots, c_j) =: t_r \in \mathcal{T}(\mathcal{O})$ and let (T_1, \dots, T_j) be the child subtrees of T , the roots of which are colored with the tuple t_r . Notice that for all $i \leq j$, T_i is an equivalence class of G/C_i^r . The algorithm will output the next equivalence class T' such that $\tilde{C}(r(T')) = t_r$ if there remains one (same color tuple at the root), or it will output the first solution such that $\tilde{C}(r(T')) = t_{r+1} \in \mathcal{T}(\mathcal{O})$ otherwise (the next color tuple at the root).

To find the next solution corresponding to the root color tuple t_r , the algorithm will replace recursively T_j by its successor T'_j w.r.t. \prec if there exists one. We obtain the solution T' whose subtrees are $(T_1, \dots, T_{j-1}, T'_j)$ which is by definition the successor of T in \prec whenever T'_j is the successor of T_j . If T_j has no successor (that is, if it is the last one), we replace if possible T_{j-1} by its successor T'_{j-1} and we replace T_j by the smallest admissible solution (i.e., the successor of \emptyset). In general, we select at each step the greatest index ℓ such that T_ℓ has a successor w.r.t. \prec , we replace it by its successor T'_ℓ and we take the smallest admissible solution for every $\ell < i \leq j$.

Without further care, the above described procedure would output solutions whose child subtrees (T_1, \dots, T_j) of the root correspond to the elements of the Cartesian product of $\mathbb{C}(G/C_i^r)$, $i \leq j$. However, while it is true that if T is a solution, its child subtree T_i is an equivalence class of G/C_i^r for all $i \leq j$, the converse is not true. Indeed, not all elements of $\mathbb{C}(G/C_1^r) \times \dots \times \mathbb{C}(G/C_j^r)$ lead to an admissible solution (an example is given in the right panel of Figure 3). In order to find an admissible solution, we should guarantee that the choice of a given T_i is *compatible* with the previous choices (T_1, \dots, T_{i-1}) . This is done by selecting the subset of OR^+ nodes $\mathcal{O}_i \subseteq C_i^r$ that are compatible with (T_1, \dots, T_{i-1}) (see Definition 5 below). An admissible choice of T_i will then be any equivalence class of G/\mathcal{O}_i . The two key properties are that the set \mathcal{O}_i can be easily computed, and that it is never empty, i.e., there is always a choice for T_i that is compatible with the previous choices of (T_1, \dots, T_{i-1}) (there is at least one choice that corresponds to the current solution). Notice that if the latter were not true, the algorithm would not have a polynomial delay complexity since we may spend exponential time without reaching a final solution. With this property, we are guaranteed that we can always extend a partial tuple (T_1, \dots, T_i) until we reach a complete tuple (T_1, \dots, T_j) that will form a solution.

Compatible nodes

Given a set of OR^+ nodes \mathcal{O} all colored with the same color c and a tree T of $\mathbb{C}(G/\mathcal{O})$, we denote by $r(\pi^{-1}(T)) := \{r(S) : S \in \pi^{-1}(T)\}$ the subset of OR^+ nodes of \mathcal{O} , each one of which is the root of a solution subtree of class T . The following definition formalizes the notion of compatible nodes mentioned previously.

► **Definition 5.** Let \mathcal{O} be a set of OR^+ nodes of color c , $t_r = (c_1, \dots, c_j) \in \mathcal{T}(\mathcal{O})$, and let T_1, \dots, T_k , with $k < j$, be respectively equivalence classes of $\mathbb{C}(G/C_i^r)$ for all $i \leq k$. We say that a node $o \in C_{k+1}^r$ is compatible with (T_1, \dots, T_k) if there exists an AND-node $x \in Ch^r(\mathcal{O})$ such that o is a child of x and such that $r(\pi^{-1}(T_i))$ contains a child of x for all $i \leq k$.

2.3.3 Analysis

► **Proposition 6.** Let T be an equivalence class of G/\mathcal{O} for a set \mathcal{O} of OR^+ nodes of G , all colored with the same color c . Then, the function *Next* of Algorithm 1 is such that:

1. *Next* (\emptyset, \mathcal{O}) returns the smallest equivalence class of G/\mathcal{O} w.r.t. \prec .
2. *Next* (T, \mathcal{O}) returns the equivalence class of G/\mathcal{O} that follows T w.r.t. \prec .
3. if T is the last equivalence class of G/\mathcal{O} , *Next* (T, \mathcal{O}) returns \perp .

Due to space constraints, some proofs are omitted and can be found in the full version of this paper [36].

► **Theorem 7.** Given an e -colored ad-AND/OR graph G , the set $\mathbb{C}(G)$ can be enumerated with delay $O(n \cdot s)$ where n is the number of nodes of G and s is the maximum size of a solution.

Proof. To enumerate $\mathbb{C}(G)$, we first split the start nodes of G into sets S_0, \dots, S_k according to their colors. For each set S_i , starting with $T = \emptyset$, we repeatedly assign *Next* (T, S_i) to T and output it until $T = \perp$. By Proposition 6, this guarantees that we output every solution of $\mathbb{C}(G/S_i)$ exactly once. Since any solution of $\mathbb{C}(G)$ belongs to $\mathbb{C}(G/S_i)$ for a given $i \leq k$, every solution of $\mathbb{C}(G)$ will be outputted exactly once.

For the complexity, notice that at most one recursive call is performed by the node of the next solution. More precisely, if *Next* $(T, \mathcal{O}) = T'$, there will be exactly one recursive call per node in T' that is not in T , and thus at most s recursive calls will be performed.

In each recursive call, both the set $\mathcal{T}(\mathcal{O})$ and the partition $\{C_i^r\}_{i \leq j}$ of grandchildren of \mathcal{O} can be computed in $O(n)$ time. It remains to show that the sets of compatible nodes \mathcal{O}_i , $i \leq j$, can be computed in $O(n)$ time in total which will conclude the proof. To do this, we should be able to compute the sets $r(\pi^{-1}(T_i))$ for all $i \leq j$. If *Next* $(T, \mathcal{O}) = T'$, the easiest way is to return the set $r(\pi^{-1}(T'))$ together with T' when the call *Next* (T, \mathcal{O}) returns. This could be done by observing that if $T \in \mathbb{C}(G/\mathcal{O})$ where \mathcal{O} is a set of goal nodes all having the same color c , then $r(\pi^{-1}(T)) = \mathcal{O}$, and if T has child subtrees T_1, \dots, T_j then $r(\pi^{-1}(T))$ is the set of nodes of \mathcal{O} that has at least an AND-child x such that the children of x contain exactly one node in $r(\pi^{-1}(T_i))$ for each $i \leq j$, which can be found in $O(n)$ time. Thus only $O(n)$ time is necessary at each recursive call to return $r(\pi^{-1}(T'))$ in addition to T' . ◀

3 Application to dynamic programming

3.1 A formalism for tree-sequential dynamic programming

Since its introduction by Karp and Held [21], *monotone sequential decision processes* (mSDP) have been the classical model for problems solvable by dynamic programming (DP). This formalism is based on finite-state automata. The solutions of DP-problems are thus equivalent

to languages of regular expressions, or to paths in directed graphs. It is known that Bellman's *principle of optimality* [5] also applies to problems for which the solutions are not sequential but *tree-like* [9]. Various generalizations have been proposed to characterize broader classes of problems solvable by DP or DP-like techniques [11, 19]. In this paper, we consider a framework which is the immediate generalization of the mSDP model, i.e., generalizing finite automata (regular expressions, paths in DAGs) to finite tree automata (regular tree grammars, solution trees of general AND/OR graphs). Further generalizations exist (from trees to graphs of treewidth > 1); the collection of these methods is known as *Non-serial dynamic programming* [6].

In this model, a tree-sequential problem can be specified by a finite (bottom-up) tree automaton $A = (Q, \Sigma, \delta, q_0, Q_F)$, where Q is a finite set of states, Σ is a ranked alphabet, δ is a set of transition rules of the form (q_1, \dots, q_n, a, q) where $q_1, \dots, q_n, q \in Q$ and $a \in \Sigma$, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is a set of final states. The problem specification also includes a cost function. The set $L(A)$ of trees accepted by the tree automaton A defines the set of feasible solutions. The minimization problem seeks to minimize the cost function over the set $L(A)$ of feasible solutions.

We will consider the simple case of a positive additive cost function that always equals zero in the initial state. An additive cost function can be defined via an incremental cost function $I: Q^* \times \Sigma \rightarrow \mathbb{R}$, where Q^* consists of tuples of states in Q of the form (q_1, \dots, q_n) . $I(q_1, \dots, q_n, a)$ can be viewed as the cost of attaching n child subtrees to a new root of symbol a . While it might seem restrictive to require an additive structure on the cost function, this simple case does cover many important problems admitting a DP-algorithm, for instance, Travelling Salesman [4, 18], Knapsack [22], or Levenshtein distance [34].

In this case, the answer of the minimization problem can be shown to be equal to $\min_{q \in Q_F} D(q)$, where $D: Q \rightarrow \mathbb{R}_{\geq 0}$ is defined by the following recurrence equations:

$$\begin{aligned} D(q_0) &= 0, \\ \text{for } q \neq q_0, \quad D(q) &= \min_{(q_1, \dots, q_n, a, q) \in \delta} \sum_{1 \leq i \leq n} D(q_i) + I(q_1, \dots, q_n, a). \end{aligned} \quad (1)$$

A *dynamic programming algorithm* for the minimization problem corresponds to an algorithm that computes D ; the function D is commonly called a *dynamic programming table* (a DP-tabled, also called a DP-array, or a DP-matrix). Needless to say, such an algorithm does not exist in general for given arbitrary tree automata and cost functions [20].

Using an algebraic approach, Gnesi and Montanari [16] have shown that solving the functional Equation (1) corresponds to finding the solution subtrees of a general AND/OR graph. An important special case in which DP-algorithms exist is when the underlying AND/OR graph is acyclic.

When a fixed tree is given as an input to the problem, the underlying AND/OR graph is acyclic and decomposable (that is, it is an ad-AND/OR graph). Such problems are hence naturally solvable by DP-algorithms. These algorithms are known in folklore under the name *Dynamic programming on a tree*. Many graph-theoretical problems (e.g., maximum matching, longest path) can be solved optimally on trees by DP-algorithms. Numerous real-world applications also rely on DP-algorithms on trees; examples can be found, for instance, in Data Science [30], Computer Vision [14, 33], and Computational Biology [3, 13].

Explicit construction of the ad-AND/OR graph for DP on a fixed tree

Due to its usefulness for the examples that we will develop next, in the case of DP on a fixed tree, an explicit construction of the ad-AND/OR graph from Equation (1) is described below. The construction is done in two steps. In the first step, we build a graph in which every node

retains an additional attribute, its *value*, and every OR^+ node is labeled by a state $q \in Q$. In the second step, we *prune* the graph by removing nodes that do not yield optimal values.

1. For each $(q_0, a, q) \in \delta$, create a goal node of value 0 labeled by q . Then, for each $q \neq q_0$ in post-order,
 - i. For each $(q_1, \dots, q_n, a, q) \in \delta$, create an AND node, connect it to the n OR^+ nodes labeled by q_1, \dots, q_n . Its value is equal to the sum of the values of its children, plus $I(q_1, \dots, q_n, a)$.
 - ii. Create a single OR node, connect it to every AND node created in the previous step. Its label is q , and its value is the minimum of the values of its children.
2. For each $q \in Q_F$, remove the OR node labeled by q unless its value is equal to $\min_{q \in Q_F} G(q)$. For each OR node s , remove the arc to its AND-child node s_i if the value of s_i is not equal to the value of s . Finally, remove recursively all AND nodes without incoming arcs.

3.2 Examples

3.2.1 Optimal tree coloring problem

Description

A prototypical problem that fits into the framework of DP on a tree is **OPTIMAL TREE COLORING**, that is, finding an optimal node-coloring of the input tree. Many problems of practical interest reduce to **OPTIMAL TREE COLORING**; three concrete examples are given later in this section.

If T is the input (rooted, ordered) tree and C is the set of colors, such a problem seeks a coloring $\phi: V(T) \rightarrow C$ that minimizes the cost function. There can be many constraints on the coloring function: some nodes of T may be forced to have a certain color, the possible colors of a node may depend on the colors of its descendants. In our tree-sequential dynamic programming formalism, a tree automaton and a cost function are given as part of the input. The tree automaton defines the set $L(A)$ of feasible coloring functions satisfying all those constraints. A state q can be interpreted as a colored subtree of T with a particular root color; the unique initial state is an empty coloring and transitions into a colored leaf of T ; a final state corresponds to a fully colored T with a particular root color. A commonly used form of cost functions considers the (possibly weighted) sum over the edges of the tree of the cost of putting two colors on each end of an edge, that is, an incremental cost function I of the form $I(q_1, \dots, q_n, a, p) = \sum_{1 \leq i \leq n} p(a_i, a)$ where a_i is the color of the root of the subtree in state q_i and $p: C^2 \rightarrow \mathbb{R}_{\geq 0}$ is a function that gives the cost of putting two colors at each end of an edge.

Equivalence relations on the set of solutions

A possible strategy to define equivalence classes on the solution space of the **OPTIMAL TREE COLORING** problem is to consider some colors to be locally equivalent on a node. In practical applications (see the next section), the space of colors can be quite large. Even though the precise colors of each node are necessary for correctly computing the cost function, when the solutions are analyzed by a human expert, it can be desirable to omit the colors and just look at whether the color of a node belongs to some group of colors. Therefore, this kind of equivalence relations is natural in many situations. Our Definition 4 of equivalence classes of an e-colored ad-AND/OR graph deals exactly with equivalence relations of this type.

Let e be a function that maps a color c to its “color group” $e(c)$. Two solutions of the OPTIMAL TREE COLORING problem $\phi_1, \phi_2: V(T) \rightarrow C$ are said to be *equivalent* if $\forall u \in V(T)$, $e(\phi_1(u)) = e(\phi_2(u))$. Let G be the ad-AND/OR graph associated with this instance. For each OR^+ node s of G labeled with the state q , where q is interpreted as a colored subtree of T with a particular root color c , color the node s with $e(c)$. Then G is e -colored and $\mathbb{C}(G)$ corresponds to the set of equivalence classes of the solutions of the instance. Notice that the constraint we had on the e -coloring of an ad-AND/OR graph is naturally satisfied by any meaningful function e because in a DP setting we only consider ordered trees: the i -th and j -th children of a node of T cannot be in the same color group unless $i = j$.

Concrete examples of tree coloring problems

Example 1. The first example is related to the alignment of gene sequences on a phylogenetic tree [31]. The input is a tree T , a set Σ of letters (DNA alphabet or protein alphabet), a function that labels each leaf node of T with a letter, and a distance function $d: \Sigma^2 \rightarrow \mathbb{R}_{\geq 0}$ between two letters. The goal is to extend the leaf labeling to a full labeling $\phi: V(T) \rightarrow \Sigma$ such that the sum of the distances over the edges of T is minimized. Defining equivalence relations of the solutions based on a grouping of the letters is uncontrived in this problem: for the DNA or protein alphabet, the letters can be subdivided into structurally similar groups.

Example 2. The FREQUENCY ASSIGNMENT problems are a family of problems that naturally arise in telecommunication networks, and that have been extensively studied in graph theory as a generalization of graph coloring known as the T-coloring problem [17, 29, 32]. In the variant called the list T-coloring, the input is a graph G representing the interference between radio stations, a set C of colors, a function S that gives for each vertex $v \in V(G)$ a set $S(v) \subseteq C$ of colors (possible frequencies for a station), and a set $T \subseteq C^2$ of forbidden pairs of colors (interfering frequencies). The goal is to find a coloring $\phi: V(G) \rightarrow C$ such that a $\forall v \in V(G)$, $\phi(v) \in S(v)$, and $\forall (u, v) \in E(G)$, $(\phi(u), \phi(v)) \notin T$. While this problem is hard in general, it can be solved by DP when the underlying graph is a tree. In this case, we can enumerate colorings of the input trees without any forbidden pair of colors on the edges. Defining equivalence relations by grouping some of the colors together (similar frequencies) can be a practical way of reducing the size of the output.

Example 3. The TREE RECONCILIATION problem is the main method for analyzing the co-evolution of two sets of species, the hosts and their parasites [28]. The input are two phylogenetic trees H, P (of the hosts and of the parasites, respectively), together with a mapping $\phi_0: \text{Leaves}(P) \rightarrow \text{Leaves}(H)$ that reflects the present-day parasite infections. What needs to be enumerated are then all past associations, that is, all mappings of the non-leaf nodes of the parasite tree to the nodes of the host tree that optimize a function which overall represents the sum of the number of different possible “events” weighted by the inverse of their estimated probability. The number of optimal solutions is often huge and, by applying our Algorithm 1, the enumeration of biologically inspired equivalence classes have allowed a significant reduction (in some cases from 10^{42} to only 96 classes) of the size of the output while still preserving the important biological information (see [35]).

3.2.2 Dynamic programming on tree decomposition of a graph

Many graph problems can be solved in polynomial time with a dynamic programming algorithm when the input graph has bounded treewidth (see for example [8]). The underlying idea is that, given a tree decomposition of a graph, the dynamic programming algorithm

traverses the nodes (bags) of the decomposition and consecutively solves the respective sub-problems. For vertex subset optimization problems, given a bag X , a dynamic programming algorithm generally computes for each $Z \subseteq X$ the optimal solution of the sub-problem whose intersection with X is Z . In this context, we could define two solutions to be equivalent if they intersect each bag of the decomposition in an “equivalent” way. The equivalence relation on the solutions is then defined by an equivalence relation over the subsets of each bag, and two solutions S_1 and S_2 are equivalent if for all bags X of the decomposition, $S_1 \cap X$ is equivalent to $S_2 \cap X$.

One of the simplest examples is to consider that all the nonempty subsets of vertices of a bag are equivalent. Thus, what we are interested in is whether a solution “hits” a bag (i.e., whether it has a nonempty intersection with the vertices in the bag). Consequently, two solutions would be considered equivalent if they hit the same bags.

We can also consider two subsets of a bag to be equivalent if they have the same size. In this case, two solutions would be equivalent if each bag contains the same number of vertices in the two solutions.

We believe that considering solutions in the way they are distributed along the tree decomposition of a graph could give a good overview of the diversity of the solution space.

4 Conclusion and perspectives

In this paper, we provide a general framework for the enumeration of equivalence classes of solutions in polynomial delay for a wide variety of contexts. This work opens a door to different research directions.

It would be interesting to ask whether we can efficiently enumerate groups of solutions that result from classical clustering procedures, or one representative per group. Moreover, in this paper we heavily rely on the decomposability property of the structure of the solution space. It remains open whether the problem of enumerating equivalence classes is hard without this restriction.

References

- 1 Steen A. Andersson, David Madigan, and Michael D. Perlman. A characterization of markov equivalence classes for acyclic digraphs. *Annals of Statistics*, 25(2):505–541, April 1997. doi:10.7916/D8280JSB.
- 2 Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’11, page 781–792, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989323.1989405.
- 3 Mukul S. Bansal, Eric J. Alm, and Manolis Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012. doi:10.1093/bioinformatics/bts225.
- 4 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.
- 5 Richard Bellman. *Dynamic Programming*. Dover Books on Computer Science. Dover Publications, 2013.
- 6 Umberto Bertele and Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, Inc., USA, 1972.
- 7 Anselm Blumer, Janet A. Blumer, David H. Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, July 1987. doi:10.1145/28869.28873.

- 8 Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming*, pages 105–118, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. doi:10.1007/3-540-19488-6_110.
- 9 Pierre E. Bonzon. Necessary and sufficient conditions for dynamic programming of combinatorial type. *Journal of the ACM*, 17:675–682, 1970. doi:10.1145/321607.321616.
- 10 Marília D.V. Braga, Marie-France Sagot, Celine Scornavacca, and Eric Tannier. Exploring the solution space of sorting by reversals, with experiments and an application to evolution. *IEEE/ACM transactions on computational biology and bioinformatics*, 5 3:348–56, 2008. doi:10.1109/TCBB.2008.16.
- 11 Joshua Buresh-Oppenheim, Sashka Davis, and Russell Impagliazzo. A stronger model of dynamic programming algorithms. *Algorithmica*, 60:938–968, August 2011. doi:10.1007/s00453-009-9385-1.
- 12 Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171(2):73–106, 2007. doi:10.1016/j.artint.2006.11.003.
- 13 Beatrice Donati, Christian Baudet, Blerina Sinimeri, Pierluigi Crescenzi, and Marie-France Sagot. EUCALYPT: efficient tree reconciliation enumerator. *Algorithms for Molecular Biology*, 10(1):3, 2015. doi:10.1186/s13015-014-0031-3.
- 14 Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, January 2005. doi:10.1023/B:VISI.0000042934.15159.49.
- 15 Henning Fernau, Petr A. Golovach, and Marie-France Sagot. Algorithmic Enumeration: Output-sensitive, Input-Sensitive, Parameterized, Approximative (Dagstuhl Seminar 18421). *Dagstuhl Reports*, 8(10):63–86, 2019. doi:10.4230/DagRep.8.10.63.
- 16 Stefania Gnesi, Ugo Montanari, and Alberto Martelli. Dynamic programming as graph searching: An algebraic approach. *Journal of the ACM*, 28(4):737–751, 1981. doi:10.1145/322276.322285.
- 17 William K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980. doi:10.1109/PROC.1980.11899.
- 18 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. doi:10.1137/0110015.
- 19 Paul Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the ACM*, 36(1):97–128, January 1989. doi:10.1145/58562.59304.
- 20 Toshihide Ibaraki. Classes of discrete optimization problems and their decision problems. *Journal of Computer and System Sciences*, 8(1):84–116, 1974. doi:10.1016/S0022-0000(74)80024-3.
- 21 Richard M. Karp and Michael Held. Finite-state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3):693–718, 1967. doi:10.1137/0115060.
- 22 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Basic Algorithmic Concepts*, pages 15–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-24777-7_2.
- 23 Alberto Martelli and Ugo Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039, 1978. doi:10.1145/359657.359664.
- 24 Cristian Molinaro, Amy Sliva, and Vs S. Subrahmanian. Super-solutions: Succinctly representing solutions in abductive annotated probabilistic temporal logic. *ACM Transactions on Computational Logic*, 15(3), July 2014. doi:10.1145/2627354.
- 25 Katherine Morrison. An enumeration of the equivalence classes of self-dual matrix codes. *Advances in Mathematics of Communications*, 9:415, 2015. doi:10.3934/amc.2015.9.415.
- 26 Kazuyuki Narisawa, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Efficient computation of substring equivalence classes with suffix arrays. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 340–351, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/s00453-016-0178-z.

- 27 Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag Berlin Heidelberg, Tioga, Palo Alto, CA, 1982.
- 28 Roderic D. M. Page. *Tangled trees: phylogeny, cospeciation, and coevolution*. The University of Chicago Press, 2003.
- 29 Fred S. Roberts. T-colorings of graphs: recent results and open problems. *Discrete Mathematics*, 93(2):229–245, 1991. doi:10.1016/0012-365X(91)90258-4.
- 30 Lior Rokach and Oded Z. Maimon. *Data Mining with Decision Trees: Theory and Applications*. Series in machine perception and artificial intelligence. World Scientific, 2008. doi:10.1142/9097.
- 31 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. doi:10.1137/0128004.
- 32 Barry A. Tesman. List t-colorings of graphs. *Discrete Applied Mathematics*, 45(3):277–289, 1993. doi:10.1016/0166-218X(93)90015-G.
- 33 Olga Veksler. Stereo correspondence by dynamic programming on a tree. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 384–390, 2005. doi:10.1109/CVPR.2005.334.
- 34 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974. doi:10.1145/321796.321811.
- 35 Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinimeri. Cappybara: equivalence class enumeration of cophylogeny event-based reconciliations. *Bioinformatics*, 36(14):4197–4199, 2020. doi:10.1093/bioinformatics/btaa498.
- 36 Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinimeri. A general framework for enumerating equivalence classes of solutions, 2021. arXiv:2004.12143.

Engineering MultiQueues: Fast Relaxed Concurrent Priority Queues

Marvin Williams ✉

Karlsruhe Institute of Technology, Germany

Peter Sanders ✉

Karlsruhe Institute of Technology, Germany

Roman Dementiev ✉

Intel Deutschland GmbH, München, Germany

Abstract

Priority queues with parallel access are an attractive data structure for applications like prioritized online scheduling, discrete event simulation, or greedy algorithms. However, a classical priority queue constitutes a severe bottleneck in this context, leading to very small throughput. Hence, there has been significant interest in concurrent priority queues with relaxed semantics. We investigate the complementary quality criteria *rank error* (how close are deleted elements to the global minimum) and *delay* (for each element x , how many elements with lower priority are deleted before x). In this paper, we introduce *MultiQueues* as a natural approach to relaxed priority queues based on multiple sequential priority queues. Their naturally high theoretical scalability is further enhanced by using three orthogonal ways of batching operations on the sequential queues. Experiments indicate that MultiQueues present a very good performance–quality tradeoff and considerably outperform competing approaches in at least one of these aspects.

We employ a seemingly paradoxical technique of “wait-free locking” that might be of more general interest to convert sequential data structures to relaxed concurrent data structures.

2012 ACM Subject Classification Computing methodologies → Concurrent computing methodologies

Keywords and phrases concurrent data structure, priority queues, randomized algorithms, wait-free locking

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.81

Related Version This paper continues a previous paper that only appeared as a preprint and SPAA brief announcement listed below. New contributions compared to that are measures for better locality, more analysis of element delays, and more realistic experiments.

Preprint: <https://arxiv.org/abs/1411.1209> [20]

SPAA brief announcement: <https://dl.acm.org/doi/10.1145/2755573.2755616> [19]

Supplementary Material The source code of our MultiQueue implementation as well as the code for the experimental evaluation can be found as follows:

Software: <https://github.com/marvinwilliams/multiqueue>

Software (Experimental Evaluation): https://github.com/marvinwilliams/multiqueue_experiments

Software (Archived Source Code and Experimental Data): https://algo2.iti.kit.edu/williams/esa21_multiqueues/

Funding This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 882500).



© Marvin Williams, Peter Sanders, and Roman Dementiev;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 81; pp. 81:1–81:17

Leibniz International Proceedings in Informatics



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

1 Introduction

Priority queues (PQs) are a fundamental data structure for many applications. They manage a set of elements and support operations for efficiently inserting elements and deleting the smallest element (*deleteMin*). Whenever we have to dynamically reorder operations performed by an algorithm, PQs can turn out to be useful. Examples include job scheduling, graph algorithms for shortest paths and minimum spanning trees, discrete event simulation, best first branch-and-bound, and other best first heuristics.

On modern parallel hardware, we often have the situation that p parallel threads want to access the PQ concurrently both to insert and to delete elements. This is problematic for several reasons. First of all, even the semantics of a parallel PQ is unclear. The classical notion of serializability is not only expensive to achieve but also not very useful from an application point of view. For example, in a branch-and-bound application, a serializable parallel PQ could arbitrarily postpone insertion and corresponding deletion of a search tree node on the path leading to the eventual solution. This makes the application arbitrarily slower than the sequential one. The *ideal* semantics of a concurrent PQ (CPQ) would be that any element for which an insertion has started becomes visible instantaneously for deletion from any other thread. This is unattainable for fundamental physical reasons but can serve as a basis for defining the quality of relaxed priority queues (RPQs). In Section 2 we use this approach to define the complementary notions of the *rank error* of deleted elements and the *delay* of elements that are overtaken by larger elements. After discussing related work in Section 3, Section 4 describes the main contribution of this paper: MultiQueues are a simple approach to CPQs based on $c \cdot p$ sequential PQs (SPQs) for some constant $c > 1$. Insertions go to a random SPQ so that each of them contains a representative sample of the globally available elements. Deleted elements are the smaller of the minima of *two* randomly chosen SPQs. By choosing two rather than one queue, fluctuations in the distribution of queue elements are stabilized. Consistency is maintained by locking queues that are changed. See Figure 1 for an example. Since there are more queues than threads, no thread ever has to wait for a lock. Thus, we achieve a lock-free (and even wait-free) algorithm despite using locks which is an interesting feature of MultiQueues.

Although MultiQueues scale better than competitors both in theory and practice, they have the practical problem that they lack cache locality – in each operation, a thread accesses several cache lines from randomly chosen SPQs which are rarely reused but usually cause cache invalidation costs later. We therefore introduce three orthogonal measures for improving locality: (1) Insertion and deletion *buffers* ensure that most operations access only a small number of cache lines. (2) We use sequential queues that allow *bulk access* in a more cache-efficient way than using several single-element operations. (3) Threads optionally *stick* to the same set of queues for several consecutive operations. In the analysis (Section 4.5), we show that MultiQueue operations are almost as fast as their sequential counterparts – scaling linearly with the number of threads. We are also able to analyze rank error and delay

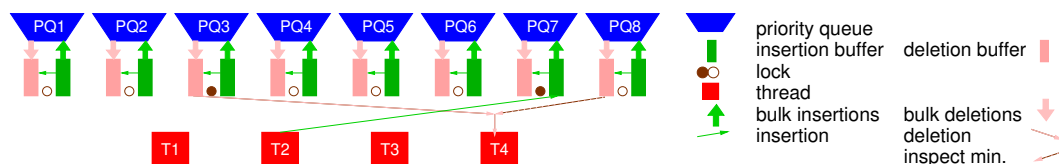


Figure 1 A MultiQueue with 8 sequential queues and 4 threads. Thread T2 currently locks PQ7 to insert an element. T4 inspected PQ3 and PQ8 and now locks PQ8 to remove its smallest element.

under moderately simplifying assumptions – it is linear in expectation and $O(p \log p)$ with high probability. Section 5 summarizes an extensive experimental evaluation of MultiQueues including a comparison with alternative approaches. Section 6 concludes the paper.

2 Preliminaries

A priority queue pq represents a set of elements. We use $n = |pq|$ for the size of the queue. Classical priority queues support the operations *insert* for inserting an element and *deleteMin* for obtaining and removing the smallest element. The most frequently used SPQ is the binary heap [29].

A *relaxed CPQ* does not require the *deleteMin* operation to return the minimum element. The *rank* of an element of a set M is its position in a sorted representation of M . A natural quality criterion for the *deleteMin* operation is the *rank error* of the returned element, i.e., the number of elements in the CPQ at the time of deletion that are smaller than the returned element. Over the entire use of the CPQ, one can look at the mean rank, the largest observed rank, or, more generally, the distribution of observed ranks. A complementary measure that has previously been largely neglected is the *delay* of a queue element x . The delay of x is the number of elements with lower priority than x that are deleted before x . This measure is important because it can be closely tied to the performance of applications. For example, consider a CPQ used in an optimization problem where the currently best queue element leads to the ultimate solution. If the CPQ from now on always remove the second best element, it has excellent rank error but will never find the solution because the best element is infinitely delayed.

It is difficult for a CPQ to consistently and efficiently maintain its exact *size*. A straightforward way to approximate the size can for example be implemented similarly to concurrent hash tables [16]. While knowing the size is not of direct importance for most applications, many of them need some kind of *termination detection*. Sequential algorithms would often terminate after they found the queue to be *empty*. Even the check for emptiness is difficult in concurrent settings since there is no way to know whether some other thread has concurrently called an insertion operation (or is about to do so). Hence, we view termination detection as a problem that has to be solved by the application. We adopt the semantics of most RPQs we are aware of that the *deleteMin* operation is allowed to fail – implying that the queue could not find any remaining elements without being able to prove that the queue is actually empty. A failed *deleteMin* has the current size of the queue as its rank error, i.e., as if ∞ was removed from the queue.

An algorithm is *wait-free* if it is guaranteed to make progress in a bounded number of steps [13]. Since we discuss randomized algorithms, we use this term also if the bound is probabilistic, i.e., if the expected number of steps is bounded.

3 Related Work

There has been considerable work on bulk parallel priority queues (BPQs) in the 1990s [8, 18, 23]. BPQs differ from CPQs in that they assume synchronized batched operation but they are still relevant as a source of ideas for asynchronous implementations. Indeed, Sanders [23] already discusses how these data structures could be made asynchronous in principle: Queue server threads periodically and collectively extract the globally smallest elements from the queue moving them into a buffer that can be accessed asynchronously. Note that within this buffer, priorities can be ignored since all buffered elements have a low

rank. Similarly, an insertion buffer keeps recently inserted elements. Moreover, the best theoretical results on BPQs give us an idea of how well RPQs should scale asymptotically. For example, Sanders' BPQ [23] removes the p smallest elements of the BPQ in time $O(\log n)$. This indicates that the worst-case rank error and delay close to *linear* in the number of threads should be achievable.

Sanders' BPQ [23] is based on the very simple idea to maintain a local SPQ on each thread and to send inserted elements to random threads. This idea is older, stemming from Karp and Zhang [14]. This approach could actually be used as an RPQ. Elements are inserted into the SPQ of a randomly chosen thread. Each thread deletes elements from its local SPQ. It is shown that this approach leads to only a constant factor more work compared to a sequential algorithm for a globally synchronized branch-and-bound application where producing and consuming elements takes constant time. Unfortunately, for a general asynchronous RPQ, the Karp-Zhang-queue [14] has limitations since slow threads could "sit" on small elements, while fast threads would busily process elements with high rank – in the worst case, the rank error could grow arbitrarily large. Our MultiQueue builds on the Karp-Zhang-queue [14], adapting it to a shared memory setting, decoupling the number of queues from the number of threads, and, most importantly, using a more robust protocol for *deleteMin*.

Many previous CPQs are based on the SkipList data structure [17]. At its bottom, the SkipList is a sorted linked list of elements. Search is accelerated by additional layers of linked lists. Each list in level i is a random sample of the list in level $i - 1$. Many previous CPQs delete the exact smallest element [26, 27, 15, 6]. This works well if there are not too many concurrent *deleteMin* operations competing for deleting the same elements. However, this inevitably results in heavy contention if very high throughput is required. The SprayList [3] reduces contention for *deleteMin* by removing not the global minimum but an element among the $O(p \log^3 p)$ smallest elements. However, for worst case inputs, insertions can still cause heavy contention. This is a fundamental problem of any data structure that attempts to maintain a single globally sorted sequence. Wimmer et al. [31] describe a RPQ for task scheduling based on a hybrid between local and global linked lists and local SPQs. Measurements in Alistarh et al. [3] indicate that this data structure does not scale as well as SprayLists – probably due to a frequently accessed central linked list. Henzinger et al. [12] give a formal specification of RPQs and mention a SkipList based implementation without giving details. Interestingly, for a relaxed FIFO-queue, the same group proposes a MultiQueue-like structure [11].

The *contention avoiding priority queue (CAPQ)* [21, 22] is based on a centralized skip-list S but switches to thread-local insertion and deletion buffers when it detects contention on S . To maintain some global view, operations will still occasionally use S . This combines high accuracy in uncontended situations with high throughput under contention. However, the quality penalty for switching to local buffers is fairly high. Assuming the centralized queue is accessed every $m \in \Theta(p)$ steps (which seems necessary to avoid contention) the paper shows that a rank error in $O(p^2)$ is guaranteed for every m -th step. Nothing can be guaranteed for the remaining fraction of $1 - 1/m$ operations.¹

For large SPQs, cache-efficient data structures are useful. Unfortunately, the best of these (e.g., [24]) are not useful for CPQs since they are only efficient in an amortized sense and locking an SPQ for an extended period of time could lead to large rank errors and delays.

¹ Such a guarantee could also be achieved with a simplistic version of MultiQueues where each thread inserts and deletes from a fixed queue except that every p steps a thread scans all queues for the globally smallest element.

4 MultiQueues

We first describe the basic MultiQueue in Section 4.1 and then refine it to improve cache locality in Sections 4.2–4.4. In Section 4.5 we provide a simplified theoretical analysis of the run time and quality of the MultiQueue. Section 4.6 discusses implementation details.

4.1 Basic MultiQueue

The basic MultiQueue data structure is an array Q of $c \cdot p$ SPQs where c is a tuning parameter and p is the number of parallel threads. Figure 1 gives an example with $c = 2$. Access to each local queue is protected by a lock flag. The *insert* operation locks a random unlocked queue $Q[i]$ and inserts the element into $Q[i]$. Figure 2 gives pseudocode. Note that this operation is wait-free since we never wait for a locked queue. Since at most p queues can be locked at any time, for $c > 1$ we will have a constant success probability. Hence, the expected time for acquiring a queue is constant. Together with the time for insertion we get expected insertion time $O(\log n)$.

An analogous implementation of *deleteMin* would lock a random unlocked queue and return its minimal element. However, the quality of this approach leaves a lot to be desired. In particular, it deteriorates not only with p but also with the queue size. One can show that the rank error grows proportional to \sqrt{n} due to random fluctuations in the number of operations addressing the individual queues. Therefore we invest slightly more effort into the *deleteMin* operation by looking at *two* random queues and deleting from the one with the smaller minimum. Figure 2 gives pseudocode for the *insert* and *deleteMin* operations. Our intuition why considering two choices may be useful stems from previous work on randomized load balancing, where it is known that placing a ball on the least loaded of two randomly chosen machines gives a maximum load that is very close to the average load independent of the number of allocated balls [4].

Even when the queue is small, cache efficiency is a major issue for MultiQueues since accessing a queue $Q[i]$ from a thread j will move the cache lines accessed by the operation into the cache of thread j . But most likely, some random other thread j' will next need that data causing not only cache misses for j' but also invalidation traffic for j .

Procedure *insert*(e)

```

repeat
   $i := \text{uniformRandom}(1..cp)$ 
  try to lock  $Q[i]$ 
until lock was successful
 $Q[i].\text{insertToSPQ}(e)$ 
unlock  $Q[i]$ 

```

Procedure *deleteMin*

```

repeat
   $i := \text{uniformRandom}(1..cp)$ 
   $j := \text{uniformRandom}(1..cp)$ 
  if  $Q[i].\text{min} > Q[j].\text{min}$  then swap  $i, j$ 
  try to lock  $Q[i]$ 
until lock was successful
 $e := Q[i].\text{deleteMinFromSPQ}$ ; unlock  $Q[i]$ 
return  $e$ 

```

■ **Figure 2** Pseudocode for basic MultiQueue *insert* and *deleteMin*. This code assumes that an empty SPQ returns ∞ as the minimum element. A return value of ∞ from *deleteMin* then indicates that no element could be found (which does not guarantee that all SPQs are empty). A practical implementation might make additional efforts to find elements when encountering empty SPQs.

Procedure insertToSPQ(e) if $D = \emptyset \vee e < \max D$ then if $ D < b$ then $D := D \cup \{e\}$; return $(e, D) := (\max D, (D \setminus \{\max D\}) \cup \{e\})$ if $ I = b$ then flush I to M $I := I \cup \{e\}$	Procedure deleteFromSPQ if $D = \emptyset$ then return ∞ // fail $e := D.deleteMin$ if $D = \emptyset$ then refill D from $M \cup I$ return e
---	--

■ **Figure 3** Pseudocode for inserting and deleting elements from an already locked sequential queue represented by the main queue M , an insertion buffer I and a deletion buffer D .

4.2 Buffering

To reduce the average number of cache lines accessed, we represent each SPQ by the main queue M , an *insertion buffer* I and a *deletion buffer* D that is organized as a sorted ring-buffer. Each buffer has a fixed capacity b . We maintain the invariant that (unless the SPQ is empty) D contains the smallest elements of $M \cup D \cup I$. This implies that **min** of the SPQ is always the first element of D .

To maintain this invariant, **inserting** an element e into an SPQ first checks whether D is empty or contains a larger element. If so, e is inserted into D . If D was not full, this finishes the insertion. Otherwise, e becomes the largest element in D . If I is full, it is flushed into the main queue. Finally, e is inserted into I . See Figure 3 for high-level pseudocode.

DeleteMin is straightforward. When D is not empty, its smallest element is removed and returned. If this empties D , it is refilled from $M \cup I$. A simple way to do this is to first flush I into M and then extract the smallest b elements from M into D . Alternatively, we can first refill D from M only and then scan through I to swap elements smaller than $\max D$ with the largest elements from D in a fashion analogous to what is done in *insertSPQ*. This has the advantage that all interactions between the buffers and M is in the form of batches of predictable size. This will be exploited in Section 4.3.

Buffering implies that the operations most often only access the buffers themselves and the lock (accessing and modifying the buffers requires the SPQ to be locked). More specifically, an insertion “usually” reads the largest element from D and then operates on I (assuming that inserted elements rarely go to D). *deleteMin* “usually” only accesses D . When buffers are flushed or refilled, a single thread performs a batch of operations on a single queue and thus can exploit whatever locality the main queue supports. In binary heaps for example, insertions exhibit high locality. Deletions at least exhibit some locality near the root, at the variable specifying the size of the queue, and at the rightmost end of the bottom layer of the tree.

4.3 Batching

To fully exploit that MultiQueues with buffering access the main queues in a bulk fashion we can use SPQs that directly support batch operations. In our prototype, we considered *merging binary heaps*. These are structured like binary heaps but each node contains k sorted elements. The heap invariant remains that nodes contain elements that are no smaller than the elements in the parent node. Insertion and deletion are also analogous to ordinary binary heaps except that compare-and-swap operations are generalized to merge-and-split.

On the one hand, merging binary heaps yield higher cache locality for bulk operations than binary heaps since all elements in each tree node can be stored consecutively and fewer nodes have to be accessed. On the other hand, they come with additional algorithmic complexity and higher worst-case access time. They are also not easily combined with a -ary heaps that are a simpler alternative to achieve somewhat higher locality than in binary heaps.

4.4 Stickiness

As a third measure to improve cache locality we introduce the concept of *stickiness*. The stickiness parameter s controls for how many consecutive operations a thread t reuses a particular local queue for its *insert* and *deleteMin* operations. After the stickiness period of one local queue ends for t or if locking the queue fails, t randomly chooses another queue to stick to. The intuition behind this protocol is that for large enough s and $c \geq 3$, the system converges to a state where threads use disjoint queues most of the time. Stickiness provides a simple mechanism to trade-off increased cache efficiency at the cost of potentially higher rank errors and increased delay.

4.5 Analysis

In this section we analyze the theoretical performance of the MultiQueue under simplifying assumptions.

4.5.1 Running Time

We analyze the asymptotic running time of the operations *insert* and *deleteMin* of the MultiQueue in a realistic asynchronous model of shared memory computing where k threads contending to write the same machine word need time $O(k)$ to perform those operations (e.g., the aCRQW model [25, Section 2.4.1]).

► **Theorem 1.** *With $c > 1$, the expected execution time of the operations *insert* and *deleteMin* is $O(1)$ plus the time for the sequential queue access.*

Proof. Whenever a thread attempts to lock a queue q , there are at most $p - 1$ locked queues. Hence the success probability is at least

$$s := 1 - \frac{p-1}{cp} \geq 1 - \frac{1}{c} \in \Omega(1).$$

Hence the expected number of attempts is

$$\sum_{i=1}^{\infty} i s (1-s)^{i-1} = \frac{1}{s} \leq \frac{1}{1-\frac{1}{c}} = \frac{c}{c-1} \in O(1).$$

The stickiness does not affect the expected number of attempts since it only dictates the first queue to attempt to lock but not subsequent attempts in case of failure. The buffers have constant size, so all operations on them are in $O(1)$. The bulk-inserting of the insertion buffer into the queue and refilling the deletion buffer amortize to the same asymptotic cost as accessing a sequential queue for each element individually. ◀

Note that the above analysis even holds when threads are blocked: In the worst case, each thread holds a lock. The overhead for another thread to avoid these locks is already accounted for in the above proof. Hence all other threads are guaranteed to make progress (in a probabilistic sense). Thus MultiQueues are probabilistically wait-free.

Using different sequential queues, we get the following bounds for the comparison based model and for integer keys:

► **Corollary 2.** *MultiQueues with binary heaps need constant average insertion time and expected time $O(\log n)$ per operation for worst case operations sequences. For integer keys in the range $0..U$, using van Emde Boas search trees [28, 7] as local queues, time $O(\log \log U)$ per operation is sufficient.*

4.5.2 Quality

Quality analysis is still a partially open problem. However, we will explain how *rank errors* and *delays* can be estimated under simplified but intuitive assumptions in the absence of stickiness.

Let us assume that all m current elements have been allocated uniformly at random to the local queues. This assumption holds if there have been no *deleteMin* operations so far and no locking attempt during *insert* failed. It is an open question whether lock contention and *deleteMin* operations invalidate this assumption in practice. Furthermore, let us assume that we choose the queues to look at for deletion randomly and no queue is currently locked.

Rank Errors

With these assumptions, the probability to delete an element e of rank i is

$$P(\text{rank} = i) = \left(1 - \frac{2}{cp}\right)^{i-1} \frac{2}{cp}$$

The first factor expresses that the $i - 1$ elements with smaller ranks are not present at the two chosen queues. The second factor is the probability that the element with rank i is *present*. Therefore, the expected rank error in the described situation is

$$\sum_{i=1}^m iP(\text{rank} = i) \leq \sum_{i \geq 1} i \left(1 - \frac{2}{cp}\right)^{i-1} \frac{2}{cp} = \frac{c}{2}p \in O(p). \quad (1)$$

The cumulative probability that the rank of e is larger than k is

$$P(\text{rank} > k) = \left(1 - \frac{2}{cp}\right)^k, \quad (2)$$

as none of the elements with a rank smaller than k must be present on the two chosen queues. $P(\text{rank} > k)$ drops to p^{-a} for $k = \frac{ca}{2}p \ln p$, i.e., with probability polynomially large in p , we have a rank error in $O(p \log p)$.

We can also give qualitative arguments on how the performed operations change the distribution of the elements. Insertions are random and hence move the system towards a random distribution of elements. Deletions tend to remove more elements from queues with small keys than from queues with large keys, thus stabilizing the system. Alistarh et al. [2, 1] confirm our conjecture for a slightly simplified process and also show that it suffices to only “sometimes” look at more than one key. On the other hand, they show that the rank error grows in an unbounded fashion if always only a single queue is considered for deletion.

Delay

The delay of an element e with rank i is equal to the number of deletions of elements with rank higher than i . Let D denote the event that a particular *deleteMin* operation delays or removes element e . Let R denote the event that e is removed. We now compute the conditional probability that e is deleted given that it is delayed or deleted by the *deleteMin* operation. By the definition of conditional probability, we have $P(R | D) = \frac{P(R \cap D)}{P(D)}$.

With $P(R \cap D) = P(\text{rank} = i) = \left(1 - \frac{2}{cp}\right)^{i-1} \frac{2}{cp}$ and $P(D) = P(\text{rank} \geq i) = \left(1 - \frac{2}{cp}\right)^{i-1}$, we get $P(R | D) = \frac{2}{cp}$. Since $P(R | D)$ is constant, the delay follows a geometric distribution with an expected value of $\frac{cp}{2}$ and values in $O(p \log p)$ with high probability.

4.6 Implementation Details

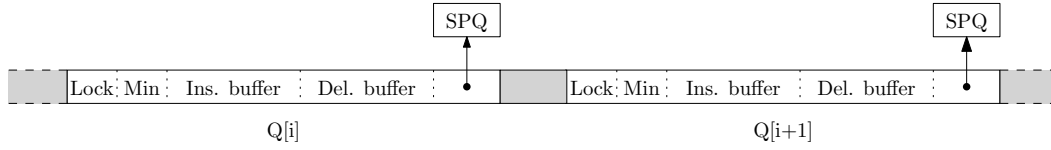


Figure 4 The array containing the lock, the buffers as well the pointer to the SPQ in each entry. The padding blocks (grey) prevent false sharing of neighboring entries.

We first give a detailed description and reasoning of the data structures used in our implementation of the MultiQueue as described above. We then show implementations of the *insert* and *deleteMin* operations.

At its core, a MultiQueue is an array Q with one entry for each local queue. Each entry contains a lock, the insertion and deletion buffers, and a pointer to the main queue itself. Moreover, the key of the minimum element in the deletion buffer is redundantly stored next to the lock. Comparing the minima of two local queues thus only involves atomic access to this key and does not access the deletion buffer, which would require locking. Note that this technique allows for slight inaccuracies, as the minimum in the deletion buffer could be different from the explicitly stored minimum key if another thread deletes the minimum from the deletion buffer during comparison. However, consistency is not impacted by this. The entries are padded and aligned to *cache lines* to prevent false sharing of neighboring entries. On systems with nonuniform memory access (NUMA) the padding is extended to *virtual memory pages* that are distributed round-robin over the NUMA-nodes. This balances memory traffic over the nodes. See Figure 4 for an illustration. Additionally, each thread stores local data such as stickiness counters. False sharing between these data structures is also avoided.

The insertion buffer is implemented as a fixed array of size b , preceded by a size counter. The deletion buffer must support efficient removal of the first element, lookup of the last element, and insertions at arbitrary positions. We chose a ring buffer that stores its size and the index of the first element upfront as the underlying data structure. Ring buffers support random lookup and removing the first element in constant time, while inserts at arbitrary positions take at most $b/2$ element moves. We implement both 8-ary heaps and k -merging binary heaps as SPQs, where k is a tuning parameter. 8-ary heaps provide the same theoretical guarantees as binary heaps but better cache locality. To achieve stable worst-case access times, our implementation allocates enough memory so that array resizes by the local queues are unlikely. Having stable worst-case access times is relevant for MultiQueues since an operation that exceptionally takes very long would lock a queue for a long time. If this queue also contains elements of low rank, their delay as well as the rank error of elements deleted in the meantime could become large. On machines with NUMA, the memory for each SPQ is allocated on the same NUMA node.

While our MultiQueue implementation can handle arbitrary element types, the implementation is designed and optimized with elements of small size in mind.

5 Experiments

We first compare our theoretical analysis of rank errors and delays with experimental results. We then perform parameter tuning and examine different implementation variants of the MultiQueue. Afterward, we compare the MultiQueue with other concurrent priority queues in

terms of quality, throughput, and scalability. For these experiments, each thread repeatedly either deletes an element from the queue or inserts a new one. This benchmark provides good insights on the maximum throughput and quality of the queues under high contention.

Queue elements are key-value pairs consisting of two 32-bit unsigned integers, where the key determines the element's priority. The queue is filled with n_0 elements prior to the measurement. Unless otherwise noted, $n_0 = 10^6$, the inserted elements are uniformly distributed in $[0, \dots, 2^{32} - 1]$, and the rank errors and delays are measured over the course of 10^7 *deleteMin* operations. To conclude this chapter, we perform a parallel single-source shortest-path (SSSP) benchmark to shed light on the performance under more realistic settings.

Experiments were conducted on two machines: Machine *A* utilizes an AMD EPYC™ 7702P 64-core processor. Each core runs at 2.0 GHz and supports two hardware threads. The system runs on Ubuntu 20.04 with Linux kernel version 5.4.0. Machine *B* is a dual socket system with an Intel® Xeon® Platinum 8368 Processor with 2.4 GHz on each socket, yielding 2×38 cores and 152 available hardware threads. The system runs on SUSE Linux Enterprise Server 15 with Linux kernel version 5.3.18. The experiments in Section 5.2 were performed on machine *A*, for the comparison experiments in Section 5.3 we additionally used machine *B*. The implementation is written in C++17 and compiled with GCC 10.2.0 with optimization level `-O3`. We use pthreads for thread management and synchronization. Each thread is pinned to a hardware thread.

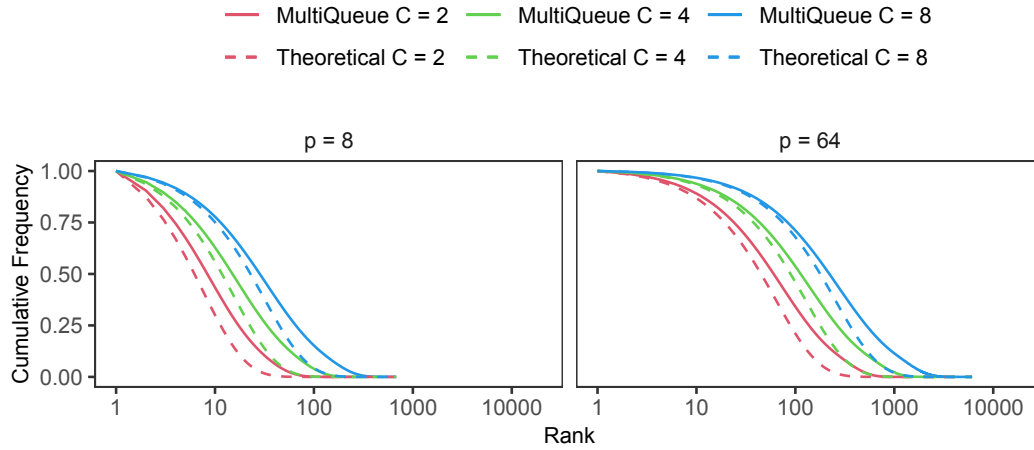
5.1 Measuring Rank Error and Delay of Relaxed PQs

Measuring the rank errors and delays in real-time imposes the practical problem that we need to know which elements are in the queue at the time of each *deleteMin* operation. We approach this problem as follows. Each thread logs its operations into a preallocated local vector. The log entries get timestamps obtained using a low-overhead high-resolution clock (we currently use the Posix `CLOCK_REALTIME`). For the evaluation, we merge these logs to one global sequence S of operations (sorted by time-stamp). S is then sequentially replayed using an augmented B+ tree (based on a `tlx::btree_map` from the `tlx` library [5]) whose leaves are the current queue elements (sorted by key). By augmenting nodes with their size, this allows determining ranks of deleted elements (and thus rank errors) in logarithmic time (e.g., [25, Section 7.5]). We further augment the interior nodes with delay counters d_v and maintain the invariant that the delay of a leaf e is the sum of the delay counters on the root- e path. When inserting an element, we can establish the invariant by setting d_e to $-\sum_v d_v$ for v on the root- e path. When performing balancing operations on the tree, we can maintain the invariant by pushing the delay counters of the manipulated nodes downward to unchanged subtrees.

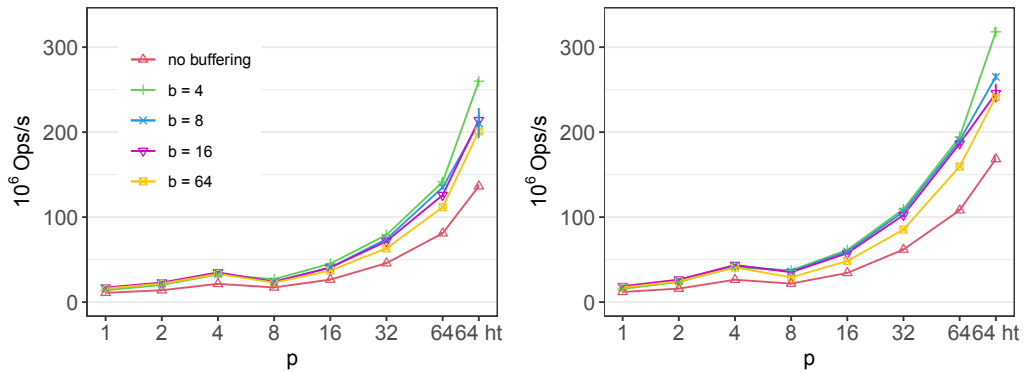
5.2 MultiQueue Parameter Tuning

The baseline MultiQueue uses 8-ary heaps as SDQs and does not use stickiness or buffers. As seen in Figure 5, the measured rank errors follow the predicted distribution closely independent of the number of threads and values for c . However, we cannot quite match the prediction in practice.² We now inspect the impact of various parameter configurations for

² The discrepancy is not due to locking since it persists when threads wait for a long time after each operation. Other sources could be noise in time measurements or inaccuracies due to our random data distribution assumption.



■ **Figure 5** The rank error distribution compared to the distribution from the theoretical analysis for $p = 8$ and $p = 64$, respectively.



■ **Figure 6** The impact of different buffer sizes to the throughput. The left plot is for $c = 2$, the right one for $c = 4$. The suffix “ht” denotes active hyper-threading.

the MultiQueue on its performance. In particular, we optimize the buffer sizes and explore different values for c and the stickiness. The throughputs are reported for a running time of 3 s and averaged over 5 runs.

Buffer Size. As Figure 6 shows, buffering increases the throughput of the MultiQueue considerably. However, the buffer size has to be chosen carefully, as too large buffers hamper the performance. This is due to the overhead of inserting elements into the deletion buffer. The difference of buffer sizes 4, 8, and 16 is negligible until hyper-threading is active, where a buffer size of 4 gains most. However, the larger the buffers, the less frequent we have to access the SPQ to perform bulk operations. Accessing the SPQ likely generates cache misses, which are especially expensive with NUMA if the SPQ is located on another NUMA node. We therefore use a buffer size of $b = 16$ for further experiments. We have observed no significant differences in rank errors and delays with or without buffering.

■ **Table 1** The throughput, average rank error and delay for $p = 64$. The number in parentheses indicates the expected rank error according to the theoretical analysis.

c	s	Throughput (MOps/s)	Avg. rank error	Avg. delay
2	1	125.9	(64) 103.1	103.1
	4	215.5	413.2	412.5
	8	271.6	900.2	896.0
	16	322.5	1853.8	1827.6
	64	411.9	7443.5	7183.2
4	1	186.5	(128) 202.6	202.6
	4	327.3	814.2	812.6
	8	405.3	1685.4	1678.0
	16	439.6	3334.9	3303.4
	64	535.3	13349.6	12870.9
8	1	222.5	(256) 405.3	405.1
	4	408.8	1598.6	1592.2
	8	488.0	3284.9	3259.5
	16	497.6	6608.4	6511.9
	64	579.8	26207.6	24902.0
16	1	244.1	(512) 811.7	810.9
	4	461.9	3216.8	3196.1
	8	542.0	6415.2	6338.3
	16	529.3	12814.4	12547.8
	64	603.3	50854.5	47695.3

Stickiness and Number of Queues. Table 1 shows measurements that suggest that rank errors and delays are not just linear in the number of queues cp but also in the stickiness s . While rank errors and delays of MultiQueues are always very similar, this is not generally true. We use the configurations $(c, s) \in \{(4, 1), (4, 4), (8, 8), (16, 8)\}$ for further benchmarks, as they provide interesting trade-offs between throughput and quality.

k -Merging Heap. Our experiments with k -binary merging heaps instead of 8-ary heaps for the SPQs indicated that merging heaps can improve the throughput for high values of s compared to 8-ary heaps. However, the impact is moderate (see Figure 7) in most cases and we stick to 8-ary heaps. Our experiments further showed that using merging heaps has no impact on the quality of the priority queue.

5.3 Comparison with Other Approaches

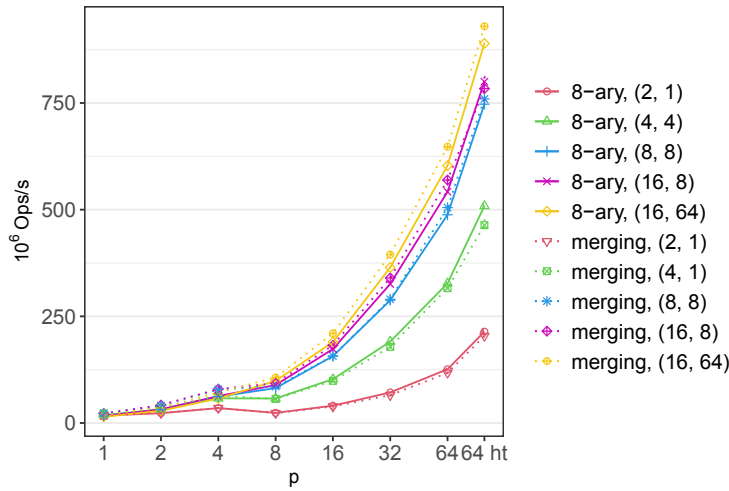
We compare the MultiQueue to the following state-of-the-art concurrent priority queues.

Linden [15] is a priority queue based on skip lists by Lindén and Jonsson. It only returns suboptimal elements in case of contention on the list head.

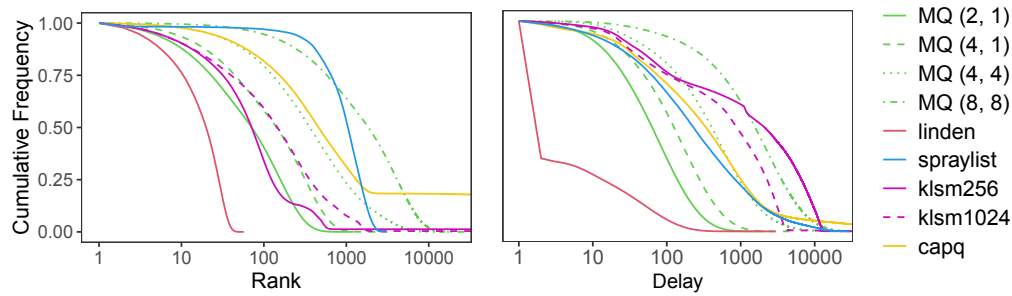
Spraylist [3] relaxes skip lists by deleting elements close to the list head to reduce contention.

k -LSM [30] combines thread-local priority queues with a relaxed shared priority queue component. We use the variants with $k = 256$ (**klsm256**) and $k = 1024$ (**klsm1024**).

CAPQ [21] dynamically detects contention to switch from using a shared skip list based priority queue to thread-local buffers.



■ **Figure 7** Throughput comparison using 8-ary heaps and k -merging heaps with $k = 16$. The numbers in parentheses are (c, s) -pairs. The suffix “ht” denotes active hyper-threading.



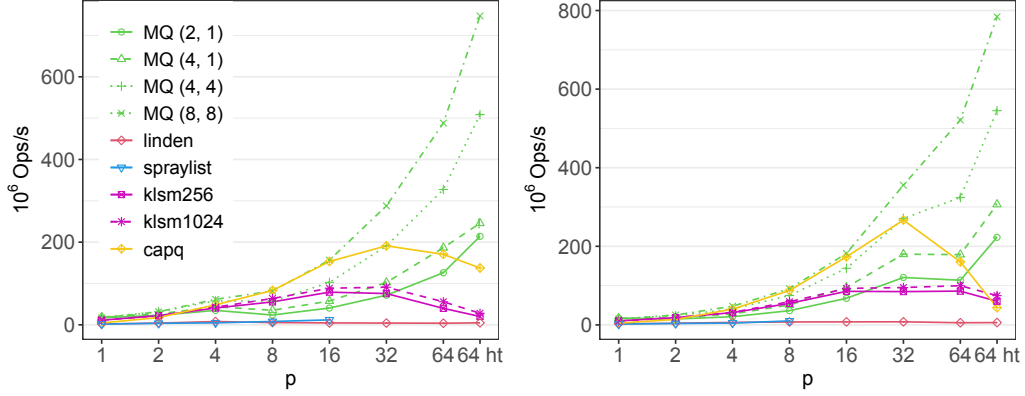
■ **Figure 8** Comparison of rank error and delay distribution for different priority queues for 10^6 *deleteMin* operations, $p = 64$.

We used the implementation found in the Github repository of the k -LSM³ for all of the above priority queues.

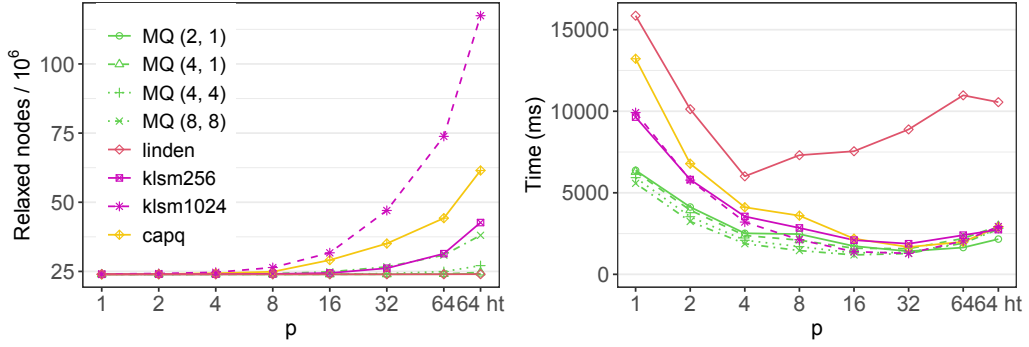
Figure 8 shows the rank error distributions and delays for 64 threads. Unsurprisingly, the Linden queue yields by far the best quality. The k -LSM achieves low rank error but high delay. For the Spraylist it is the other way around as it has high rank error but low delay. Multiqueues have similar delays and rank errors. With small values for c and s they are more accurate than all the competitors except for the Linden queue. The CAPQ exhibits quality comparable to the MultiQueue with the rather loose setting $c = s = 4$.

As seen in Figure 9, the CAPQ is among the fastest priority priority queues for up to 32 threads. Beyond that, it accesses the centralized queue so often that contention deteriorates performance. This could probably be remedied with different settings of the tuning parameters but only with a commensurate effect on even higher rank errors and delays. The very loose MultiQueue with $c = s = 8$ exhibits very good scalability. Even the higher-quality variants can take advantage of all hardware threads and eventually outperform the CAPQ. The k -LSM scales up to around 16 threads while Linden and Spraylist scale

³ <https://github.com/klmpq/klsm>



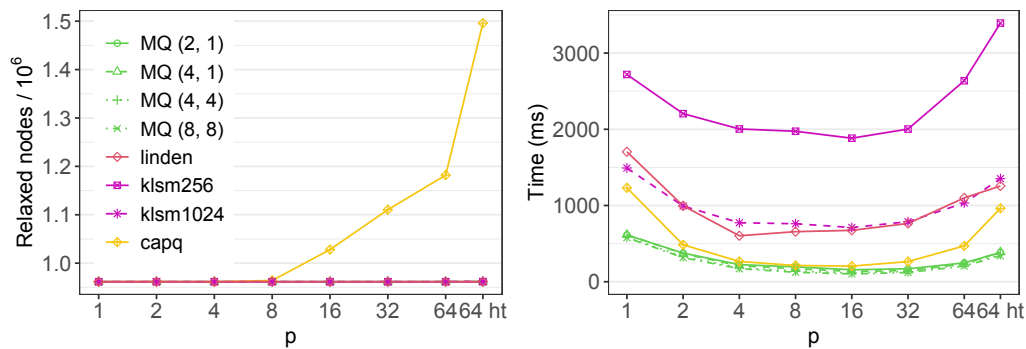
■ **Figure 9** Throughput comparison between the MultiQueue and other implementations. On the left is machine A, and on the right machine B.



■ **Figure 10** The SSSP benchmark on the *USA* graph running on machine A.

very poorly even on very few threads. The authors of the k -LSM show that the scalability of the k -LSM can be improved by selecting higher values for k such as 4096 at the cost of lower quality [10]. Machines A and B have similar performance with B having a slight advantage when all threads are used but suffering from the switch from one to two sockets. Unfortunately, the Spraylist crashed in our setup on both machines with higher thread count, so we had to exclude it from the SSSP benchmark below.

The parallel SSSP benchmark uses Dijkstra's algorithm to calculate the shortest paths from one node to all other nodes in a weighted graph. Dijkstra's algorithm has to be adapted to be used in a parallel setting with relaxed priority queues. Sagonas and Winblad use a similar benchmark for the CAPQ and describe the algorithm in greater detail [21]. The algorithm terminates as soon as the queue is empty. To detect when the queue is empty, they use the property of the CAPQ and k -LSM that if *deleteMin* fails for all threads some time after the last insertion happened, the queue must be empty. This does not hold for MultiQueues, thus we cannot rely solely on *deleteMin* to guarantee correct termination. Instead, we use a dedicated emptiness detection routine, where a thread checks c designated local queues for emptiness if it thinks that the queue is empty. The MultiQueue is empty if all threads have successfully completed this emptiness check some time after the last insertion has happened. For $c > 2$ the emptiness check is slightly more expensive than a *deleteMin* operation, but it is only executed after multiple failed *deleteMins*, so the added overhead is minuscule. We report the time to solve the SSSP problem for different thread counts as



■ **Figure 11** The SSSP benchmark on a random hyperbolic graph with 2^{20} nodes, an average degree of 16 and $\gamma = 2.3$ running on machine A.

well as the number of nodes that were extracted from the queue and then relaxed. We used real road networks and artificial random hyperbolic graphs (rhg) as benchmark instances. The road network *USA*⁴ has about 24m nodes and 58m edges. The road network *GER*⁵ has about 20m nodes and 42m edges. The weights on these graphs represent the travel time. We used a modified version of the KaGen framework [9] to generate random hyperbolic graphs with 2^{20} and 2^{22} nodes and the geometric distances as edge weights. Figure 10 and 11 give results for the *USA* road network and the rhg with 2^{20} nodes on machine A, respectively.

On the road map graphs, all the relaxed PQs considerably outperform the Linden queue. However, none of them scales particularly well beyond 16 threads with the MultiQueue variants performing best. Despite having considerably better rank errors than the CAPQ, the klsm1024 leads the algorithm to processes many more nodes on the *USA* graph. The k -LSM variants have very high delays, which indicates that the delay is a distinctive metric to measure the quality of relaxed priority queues. Rhgs paint a different picture: While only the CAPQ leads to nodes being processed more than once, both k -LSM variants are noticeably slower than both the CAPQ and MultiQueues. MultiQueues lead to very low overhead considering the extracted nodes in all our benchmarks and are very competitive at the same time. Especially in algorithms where processing the individual elements is expensive, this could be a decisive factor.

6 Conclusions and Future Work

MultiQueues are a simple and efficient approach to relaxed concurrent priority queues. They allow a transparent trade-off between throughput and quality and considerably outperform previous approaches in at least one of these aspects. An important open problem is to complete the theoretical analysis to encompass stickiness. We believe that further practical improvements could be possible by better avoiding conflicting queue accesses of the sticky variant even when few queues are used. It would also be interesting to make MultiQueues contention aware to achieve higher quality and more graceful degradation than the simple binary switch between local and global access used in the CAPQ [21].

A conceptual contribution of our paper is in introducing the quality measure of *delay* as a complement to *rank error*. We give evidence that this is important for the actual performance of applications such as shortest path search and explain how to measure it efficiently. We are

⁴ <http://users.diag.uniroma1.it/challenge9/download.shtml>

⁵ <https://i11www.itk.kit.edu/resources/roadgraphs.php>

currently having a closer look at using CPQs for branch-and-bound where it seems that the parallel performance of the applications is provably tied to the delays incurred by the CPQ. Beyond priority queues, it would be interesting to see whether our approach to “wait-free” locking can be used for other applications, e.g., for FIFOs.

References

- 1 Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z. Li, and Giorgi Nadiradze. Distributionally linearizable data structures. In *30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 133–142, New York, NY, USA, 2018. doi: 10.1145/3210377.3210411.
- 2 Dan Alistarh, Justin Kopinsky, Jerry Li, and Giorgi Nadiradze. The power of choice in priority scheduling. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 283–292, 2017.
- 3 Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The SprayList: A scalable relaxed priority queue. Technical Report MSR-TR-2014-16, Microsoft Research, September 2014. URL: <http://research.microsoft.com/apps/pubs/default.aspx?id=209108>.
- 4 P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. In *32th Annual ACM Symposium on Theory of Computing*, pages 745–754, 2000.
- 5 Timo Bingmann. TLX: Collection of sophisticated C++ data structures, algorithms, and miscellaneous helpers, 2018. , retrieved Oct. 7, 2020. URL: <https://panthema.net/tlx>.
- 6 Irina Calciu, Hammurabi Mendes, and Maurice Herlihy. The adaptive priority queue with elimination and combining. *CoRR*, abs/1408.1021, 2014. URL: <http://arxiv.org/abs/1408.1021>, arXiv:1408.1021.
- 7 R. Dementiev, L. Kettner, J. Mehnert, and P. Sanders. Engineering a sorted list data structure for 32 bit keys. In *6th Workshop on Algorithm Engineering & Experiments*, pages 142–151, New Orleans, 2004.
- 8 N. Deo and S. Prasad. Parallel heap: An optimal parallel priority queue. *The Journal of Supercomputing*, 6(1):87–98, 1992.
- 9 Daniel Funke, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS, Vancouver, BC, Canada, 2018*.
- 10 Jakob Gruber, Jesper Larsson Träff, and Martin Wimmer. Benchmarking concurrent priority queues: Performance of k-lsm and related data structures, 2016. arXiv:1603.05047.
- 11 Andreas Haas, Michael Lippautz, Thomas A Henzinger, Hannes Payer, Ana Sokolova, Christoph M Kirsch, and Ali Sezgin. Distributed queues in shared memory: multicore performance and scalability through quantitative relaxation. In *Proceedings of the ACM International Conference on Computing Frontiers*, page 17. ACM, 2013.
- 12 Thomas A Henzinger, Christoph M Kirsch, Hannes Payer, Ali Sezgin, and Ana Sokolova. Quantitative relaxation of concurrent data structures. In *ACM SIGPLAN Notices*, volume 48, pages 317–328. ACM, 2013.
- 13 M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- 14 R. M. Karp and Y. Zhang. Parallel algorithms for backtrack search and branch-and-bound. *Journal of the ACM*, 40(3):765–789, 1993.
- 15 Jonatan Lindén and Bengt Jonsson. A skiplist-based concurrent priority queue with minimal memory contention. In *Principles of Distributed Systems*, pages 206–220. Springer, 2013.
- 16 Tobias Maier, Peter Sanders, and Roman Dementiev. Concurrent hash tables: Fast and general?(!). *ACM Transactions on Parallel Computing (TOPC)*, 5, 2019.
- 17 W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

- 18 A. Ranade, S. Cheng, E. Deprit, J. Jones, and S. Shih. Parallelism and locality in priority queues. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 97–103, October 1994.
- 19 Hamza Rihani, Peter Sanders, and Roman Dementiev. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues. *CoRR*, abs/1411.1209, 2014. URL: <http://arxiv.org/abs/1411.1209>, arXiv:1411.1209.
- 20 Hamza Rihani, Peter Sanders, and Roman Dementiev. Multiqueues: Simple relaxed concurrent priority queues. In *27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 80–82, 2015. doi:10.1145/2755573.2755616.
- 21 Konstantinos Sagonas and Kjell Winblad. The contention avoiding concurrent priority queue. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 314–330. Springer, 2016.
- 22 Konstantinos Sagonas and Kjell Winblad. A contention adapting approach to concurrent ordered sets. *Journal of Parallel and Distributed Computing*, 115:1–19, 2018.
- 23 P. Sanders. Randomized priority queues for fast parallel access. *Journal Parallel and Distributed Computing, Special Issue on Parallel and Distributed Data Structures*, 49:86–97, 1998.
- 24 P. Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5, 2000.
- 25 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures – The Basic Toolbox*. Springer, 2019. doi:10.1007/978-3-540-77978-0.
- 26 Nir Shavit and Itay Lotan. Skiplist-based concurrent priority queues. In *14th Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 263–268. IEEE, 2000.
- 27 Håkan Sundell and Philippas Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. In *Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 11–20. IEEE, 2003.
- 28 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. *Information Processing Letters*, 6(3):80–82, 1977.
- 29 J. W. J. Williams. Algorithm 232 (HEAPSORT). *Communications of the ACM*, 7:347–348, 1964.
- 30 Martin Wimmer, Jakob Gruber, Jesper Larsson Träff, and Philippas Tsigas. The lock-free k-lsm relaxed priority queue. *ACM SIGPLAN Notices*, 50(8):277–278, 2015.
- 31 Martin Wimmer, Francesco Versaci, Jesper Larsson Träff, Daniel Cederman, and Philippas Tsigas. Data structures for task-based priority scheduling. In *19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 379–380, New York, NY, USA, 2014. ACM. doi:10.1145/2555243.2555278.

Evidence for Long-Tails in SLS Algorithms

Florian Wörz   

Institut für Theoretische Informatik, Universität Ulm, Germany

Jan-Hendrik Lorenz  

Institut für Theoretische Informatik, Universität Ulm, Germany

Abstract

Stochastic local search (SLS) is a successful paradigm for solving the satisfiability problem of propositional logic. A recent development in this area involves solving not the original instance, but a modified, yet logically equivalent one [23]. Empirically, this technique was found to be promising as it improves the performance of state-of-the-art SLS solvers.

Currently, there is only a shallow understanding of how this modification technique affects the runtimes of SLS solvers. Thus, we model this modification process and conduct an empirical analysis of the hardness of logically equivalent formulas. Our results are twofold. First, if the modification process is treated as a random process, a lognormal distribution perfectly characterizes the hardness; implying that the hardness is long-tailed. This means that the modification technique can be further improved by implementing an additional restart mechanism. Thus, as a second contribution, we theoretically prove that all algorithms exhibiting this long-tail property can be further improved by restarts. Consequently, all SAT solvers employing this modification technique can be enhanced.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Distribution functions

Keywords and phrases Stochastic Local Search, Runtime Distribution, Statistical Analysis, Lognormal Distribution, Long-Tailed Distribution, SAT Solving

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.82

Related Version *Full-length Version*: [arXiv:2107.00378](https://arxiv.org/abs/2107.00378) [41]

Supplementary Material See [42]:

Software (Base instances and modifications): <https://doi.org/10.5281/zenodo.4715893>

Software (Visual and statistical evaluations): <https://doi.org/10.5281/zenodo.5026180>

Funding *Florian Wörz*: Supported by the Deutsche Forschungsgemeinschaft (DFG) under project number 430150230, “Complexity measures for solving propositional formulas”.

Acknowledgements The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

1 Introduction

Although algorithms for solving the NP-complete satisfiability problem, so-called SAT solvers, are nowadays remarkably successful in solving large instances, randomized versions of these solvers often show a high variation in the runtime required to solve a fixed instance over repeated runs [16]. In the past, research on randomized algorithms often focused on studying the unsteady behavior of statistical measures like the mean, variance, or higher moments of the runtime over repeated runs of the respective algorithm. In particular, these measures are unable to capture the long-tailed behavior of difficult instances. In a different line of work [13, 15, 32], the focus has shifted to studying the runtime distributions of search algorithms, which helps to understand these methods better and draw meaningful conclusions for the design of new algorithms.



© Florian Wörz and Jan-Hendrik Lorenz;
licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 82; pp. 82:1–82:16

Leibniz International Proceedings in Informatics



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

Recently, the hybrid solver **GapSAT** [23] was introduced, combining a stochastic local search (SLS) solver with a conflict-driven clause learning (CDCL) solver. In the analysis conducted, it was empirically shown that adding new clauses is beneficial to the mean runtime (in flips) of the SLS solver **probsAT** [3] underlying the hybrid model. The authors also demonstrated that although adding new clauses can improve the mean runtime, there exist instances where adding clauses can harm the performance of SLS. This behavior is worth studying to help eliminate the risk of increasing the runtime of such procedures.

For this reason, we study the runtime (or more precisely, hardness) distribution of the procedure **Alfa**, introduced in this work, that models the addition of a set of logically equivalent clauses L to a formula F and the subsequent solving of this amended formula $F^{(1)} := F \cup L$ by an SLS solver. Our empirical evaluations show that this distribution is long-tailed. We want to stress the fact that studies on the runtime distribution of algorithms are quite sparse even though knowledge of the runtime distribution of an algorithm is extremely valuable: (1) Intuitively speaking, if the distribution is long-tailed, one knows there is a risk of ending in the tail and experiencing very long runs; simultaneously, the knowledge that the time the algorithm used thus far is in the tail of the distribution can be exploited to restart the procedure (and create a new logically equivalent instance $F^{(2)}$). We will prove this statement in a rigorous manner for all long-tailed algorithms. (2) Given the distribution of an algorithm's sequential runtime, it was shown how to predict and quantify the algorithm's expected speedup due to parallelization [1]. (3) If the distribution of hardness is known, experiments with few instances can lead to parameter estimations of the underlying distribution [13]. (4) Knowledge of the distribution can help compare competing algorithms: one can test if the difference in the means of algorithm runtimes is significant if the distributions are known [13].

1.1 Our Contributions

Our contributions consist of an empirical as well as theoretical part, specified below.

Statistical Runtime/Hardness Distribution Analysis. By conducting a plethora of experiments (total CPU time 80 years) and using several statistical tools for the analysis of empirical distributions, we conjecture that **Alfa** equipped with SLS solvers based on Schönning's Random Walk Algorithm [36], **SRWA** for short, follows a long-tailed distribution (Conjecture 8). The evidence obtained further suggests that this distribution is, in fact, lognormal (Conjecture 6). We measure the goodness-of-fit of our results over the whole domain using the χ^2 -statistic.

Restarts Are Useful For Long-Tailed Algorithms. Lorenz [22] has analyzed the lognormal and the generalized Pareto distribution for the usefulness of restarts and their optimal restart times. Given that our Strong Conjecture 6 holds, this result implies that restarts are useful for **Alfa**. We will also show that this is the case if only the Weak Conjecture 8 holds: We theoretically prove that restarts are useful for the class of algorithms exhibiting a long-tailed distribution.

1.2 Related Work and Differentiation

In [13], the authors presented empirical evidence for the fact that the distribution of the effort (more precisely, the number of consistency checks) required for backtracking algorithms to solve constraint satisfaction problems randomly generated at the 50 % satisfiable point

can be approximated by the Weibull distribution (in the satisfiable case) and the lognormal distribution (in the unsatisfiable case). These results were later extended to a wider region around the 50 % satisfiable point [32]. It should be emphasized that this study created all instances using the same generation model. This resulted in the creation of similar yet logically non-equivalent formulas. We, however, will firstly use different models to rule out any influence of the generation model and secondly generate logically equivalent modifications of a base instance (see Algorithm 1). This approach lends itself to the analysis of existing SLS solvers [23]. The major advantage is that the conducted work is not lost in the case of a restart: only the logically equivalent instance could be changed while keeping the current assignment.

In [16], the cost profiles of combinatorial search procedures were studied. The authors showed that they are often characterized by the Pareto-Lévy distribution and empirically demonstrated how rapid randomized restarts can eliminate this tail behavior. We will theoretically prove the effectiveness of restarts for the larger class of long-tailed distributions.

The paper [1] studied the solvers **Sparrow** and **CCASAT** and found that for randomly generated instances the lognormal distribution is a good fit for the runtime distributions. For this, the Kolmogorov-Smirnov statistic $\sup_{t \in \mathbb{R}} |\hat{F}_n(t) - F(t)|$ was used. Although the KS-test is very versatile, this comes with the disadvantage that its statistical power is rather low. Clearly, the KS statistic is also nearly useless in the tails of a distribution: A high relative deviation of the empirical from the theoretical cumulative distribution function in either tail results in a very small absolute deviation. It should also be remarked that the paper studies only few formulas in just two domains, 10 randomly generated and 9 crafted. Our work will address both shortcomings in this paper: The χ^2 -test gives equal importance to the goodness-of-fit over the entire support; and various instance domain models (both theoretical and applied) are considered in this paper.

► **Remark.** Unfortunately, the term heavy- or long-tailed distribution is not used consistently in the literature. We will follow [12] and use the notion given in Definition 7.

2 Preliminaries

We assume familiarity with terminologies such as Boolean variable, literal, clause, CNF formula, the SAT problem, and assignment flips in SLS solvers and refer the reader to e.g. [37]. We furthermore trust that the reader has basic knowledge of the proof system Resolution [8, 33]. *Stochastic local search* (SLS) solvers operate on complete assignments for a formula F . These solvers are started with a randomly generated complete initial assignment α_0 . If α_0 satisfies F , a solution is found. Otherwise, the SLS solver tries to find a solution by performing a random walk over the set of complete assignments for the underlying formula. A formula F *logically implies* a clause C if every complete truth assignment which satisfies F also satisfies C , for which we write $F \models C$. If L is a set of clauses we write $F \models L$ if $F \models C$ for all $C \in L$.

► **Definition 1.** Let X be a random variable for the runtime of an SLS algorithm \mathcal{A} on some input. For $t > 0$, the algorithm \mathcal{A}_t is obtained by restarting \mathcal{A} after time t if no solution was found. Restarts are useful if there is a $t > 0$ such that

$$\mathbb{E}[X_t] < \mathbb{E}[X],$$

where X_t models the runtime of \mathcal{A}_t .

► **Definition 2** ([19]). Let X be a real-valued random variable.

■ Its cumulative distribution function (cdf) is the function $F: \mathbb{R} \rightarrow [0, 1]$ with

$$F(t) := \Pr[X \leq t].$$

- Its quantile function $Q: (0, 1) \rightarrow \mathbb{R}$ is given by $Q(p) := \inf\{t \in \mathbb{R} \mid F(t) \geq p\}$.
- A non-negative, integrable function f such that $F(t) = \int_{-\infty}^t f(u) du$ is called probability density function (pdf) of X .

► **Definition 3** ([40]). An absolutely continuous, positive random variable X is (three-parameter) lognormally distributed with parameters $\sigma^2 > 0$, $\gamma > 0$, and $\mu \in \mathbb{R}$, if $\log(X - \gamma)$ is normally distributed with mean μ and variance σ^2 . In the following, we refer to σ as the shape, μ as the scale, and γ as the location parameter.

► **Definition 4.** Let X_1, \dots, X_n be independent, identically distributed real-valued random variables with realizations x_i of X_i . Then the empirical cumulative distribution function (ecdf) of the sample (x_1, \dots, x_n) is defined as

$$\hat{F}_n(t) := \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{x_i \leq t\}}, \quad t \in \mathbb{R},$$

where $\mathbb{1}_A$ is the indicator of event A .

3 Design of the Adjusted Logical Formula Algorithm Alfa

Our SLS solver **Alfa** (Adjusted logical formula algorithm) receives a satisfiable formula F as input. The algorithm then proceeds by adding to F a set L of logically generated clauses. It finally calls an SLS solver to solve the clause set $F \cup L$.

■ **Algorithm 1** Alfa acts as a base algorithm that can use different SLS algorithms.

Input: Boolean formula F , **Promise:** $F \in \text{SAT}$

Generate **randomly** a set L of clauses such that $F \models L$

Call $\text{SLS}(F \cup L)$ for some SLS solver SLS

Definition 5 is used in Algorithm 2 as a natural way to sample a set L of logically equivalent clauses with respect to a base instance F .

■ **Algorithm 2** Generation of the random set L with resolution.

Input: Boolean formula F , integer w , probability $p \in (0, 1]$, Boolean *shuffle*

foreach $R \in \text{Res}_w^*(F) \setminus F$ **do**

 | **with probability** p **do** $L := L \cup \{R\}$

if *shuffle* **then return** $\text{Shuffle}(L)$ **else return** L ;

► **Definition 5.** Let F be a clause set, and w be a positive integer. We define the operator

$$\text{Res}_w(F) := F \cup \{R \mid R \text{ is a resolvent of two clauses in } F \text{ and } |R| \leq w\}.$$

Also, we inductively define $\text{Res}_w^0(F) := F$ and

$$\text{Res}_w^{n+1}(F) := \text{Res}_w(\text{Res}_w^n(F)), \text{ for } n \geq 0.$$

Finally, we set

$$\text{Res}_w^*(F) := \bigcup_{n \geq 0} \text{Res}_w^n(F).$$

4 Empirical Evaluation

4.1 Experimental Setup, Instance Types, and Solvers Used

Hoos and Stützle [18] introduced the concept of *runtime distribution* to characterize the cdf of Las Vegas algorithms, where the runtime can vary from one execution to another, even with the same input. To obtain enough data for a fitting of such a distribution, for each base instance F we created 5000 modified instances $F^{(1)}, \dots, F^{(5000)}$ by generating resolvent sets $L^{(1)}, \dots, L^{(5000)}$ each by using Algorithm 2 with $w = 4$ and a value of p such that the expected number of resolvents being added was $\frac{1}{10}|F|$. Note that we also conducted a series of experiments to rule out the influence of p on our results. Each of these modified instances was solved 100 times, each time using a different seed. For $i = 1, \dots, 5000$ and $j = 1, \dots, 100$ we thus obtained the values $\text{flips}_S(F^{(i)}, s_j)$ indicating how many flips were used to solve the modified instance $F^{(i)}$ with solver S when using the seed s_j . Next, we calculated the mean number of flips $\text{mean}_S(F^{(i)}) := \frac{1}{100} \sum_{j=1}^{100} \text{flips}_S(F^{(i)}, s_j)$ required to solve $F^{(i)}$ with solver S whose hardness distribution we are going to analyze.

All experiments were performed on bwUniCluster 2.0 and three local servers. Sputnik [39] was used to distribute the computation and to parallelize the trials. Due to the heterogeneity of the computer setup, measured runtimes are not directly comparable to each other. Consequently, we instead measured the number of variable flips performed by the SLS solver. This is a hardware-independent performance measure with the benefit that it can also be analyzed theoretically. To give an indication of how flips relate to wall-clock time, one million flips take about one second of computing time on one of our servers. To give an idea of the computational effort involved, obtaining the data for the ecdf of a 100 variable base instance with SRWA took an average of 17,193,517 seconds (≈ 199 days) when unparallelized. This clearly prohibited examining instances having a number of variables currently being routinely solved by the state-of-the-art SLS algorithms. For the experiments the following instance types were used:

1. **Hidden Solution:** We implemented the CDC algorithm [5, 6] in [24] to generate instances with a hidden solution. For this, at the beginning, a complete assignment α is specified to ensure the generated formula's satisfiability. Then, repeatedly a randomly generated clause C is added to the formula with a weighted probability p_i depending on the number i of correct literals in C with respect to α . We included this type of instances because SLS solvers struggle to solve such instances. Experiments like these might be beneficial to find theoretical reasons for this behavior.
2. **Hidden Solution With Different Chances:** We also created sets of formulas with different underlying p_i values to rule out the influence of these.
3. **Uniform Random:** To generate uniform, random k -SAT instances with n variables and m clauses, each clause is generated by sampling k literals uniformly and independently. Using Gableske's `kcnfgen` [14], we generated formulas with $n \in \{50, 60, 70, 80, 90\}$ variables and a clause-to-variable ratio r close to the *satisfiability threshold* [29] of $r \approx 4.267$. We checked each instance with `Glucose3` [2, 11] for satisfiability until we had 5 formulas of each size.
4. **Factoring:** These formulas encode the factoring problem in the interval $\{128, \dots, 256\}$ and were generated with [10].
5. **Coloring:** These formulas assert that a graph is colorable with 3 colors. We generated these formulas, using [21], over random graphs with n vertices and $m = 2.254n$ edges in expectation, which is slightly below the *non-colorability threshold* [20]. We obtained 32 satisfiable instances in 150 variables.

Our experiments investigated leading SLS solvers where the dominating component is based on the random walk procedure proposed in [36]. In this paper, Schöning’s Random Walk Algorithm **SRWA** was introduced, which is one of the solvers we used. The **probsAT** solver family [3] is based on this approach. One of these solvers won the random track of the SAT competition 2013 [4]. Another advancement of **SRWA** was implemented as **YALSAT** [7], which won the random track of the SAT competition 2017 [17]. These performances and similarities were reasons for choosing **SRWA**, **probsAT**, and **YALSAT** as SLS solvers for this paper. The connection to **GapSAT** [23] is another case in point.

We excluded the solvers **DCCAlm** [25] and **CSCCSat** [28] (combining **FrwCB** [26] and **DCCASat** [27]) as all of these depend on heuristics (like **CC**, **BM**, **CSDvars**, **NVDvars**, **SDvars**) that ultimately reduce the probabilistic nature when choosing the next variable to flip.

For **SRWA** we conducted most of our experiments: All instance types were tested, including different change values for the generation of the hidden solution. For **probsAT**, 55 hidden solution instances with $n \in \{50, 100, 150, 200, 300, 800\}$ were used. Since **YALSAT** can be regarded as a **probsAT** derivate, we tested **YALSAT** with 10 hidden solution instances with 300 variables each.

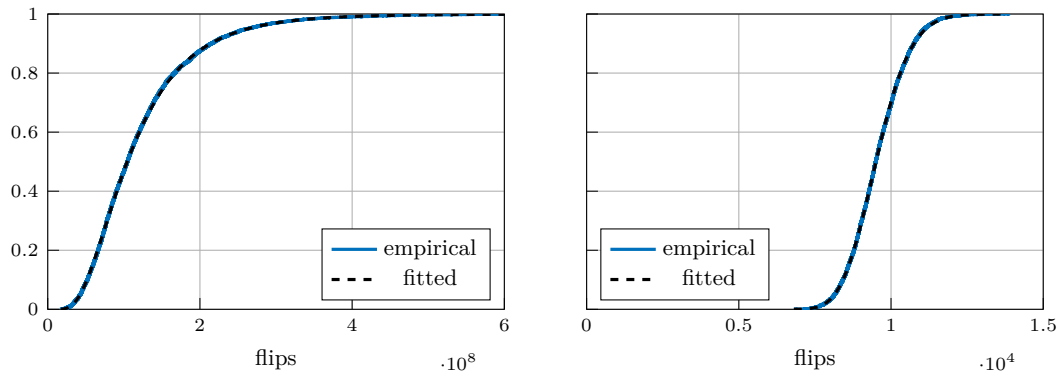
4.2 Experimental Results and Statistical Evaluation

The goal of this section is to explore the scenarios described above in more detail. We are particularly interested in how the hardness of an instance changes when logically equivalent clauses are added in the manner described above. To characterize this effect as accurately as possible, studying the ecdf is the most suitable method for this purpose. In turn, the ecdf can be described using well-known distribution types such as e.g., the normal distribution. In the following, we shall demonstrate that the three-parameter lognormal distribution, in particular, provides an exceptionally accurate description of the runtime behavior, and this is true for all considered problem domains and all solvers. The results are so compelling that we ultimately conjecture that the runtimes of **Alfa**-type algorithms all follow a lognormal distribution, regardless of the considered problem domain.

To illustrate this point, we first demonstrate our approach using two base instances. The first one is a factorization instance that was solved by **SRWA**. The second instance has a hidden solution and was solved by **probsAT**. For later reference, we refer to the first instance as *A* and to the second instance as *B*. As described above, we obtain 5000 samples for each base instance. Using these data points, we estimate the lognormal distribution’s three parameters by applying the maximum likelihood method (see [42]). After that, one can visually evaluate the suitability of the fitted lognormal distribution for describing the data. A useful method of visualizing the suitability is to plot the ecdf and the fitted cdf on the same graph.

Such a comparison is illustrated in Figure 1 for the two instances *A* and *B*. In both cases, no difference between the empirical data of the ecdf and the fitted distribution can be detected visually. In other words, the absolute error between the predicted probabilities from the fitted cdf versus the empirical probabilities from the ecdf is minuscule. Even though these are only two examples, it should be noted that these two instances are representative of the behavior of the investigated algorithms. Hardly any deviation could be observed in this plot type for all instances and all algorithms. All data is published under [42].

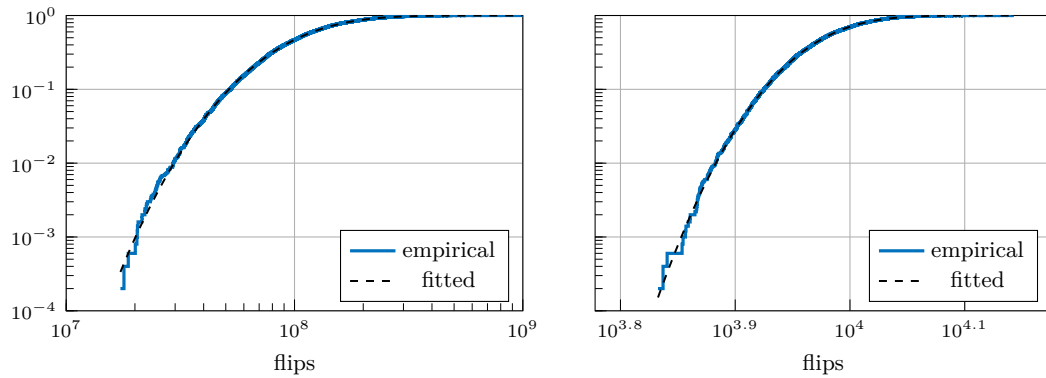
For the analysis, however, one should not confine oneself to this plot type. Although absolute errors can be observed easily, relative errors are more difficult to detect. Such a relative error may have a significant impact when used for decisions such as restarts. To illustrate this point, suppose that the true probability of a run of length ℓ is 0.0001. In contrast, the probability estimated based on a fit is 0.001. As can be seen, the absolute error



■ **Figure 1** The ecdf and fitted cdf of the hardness distribution of instance *A* (left) and *B* (right).

of 0.0009 is small, whereas the relative error of 10 is large. If one were to perform restarts after ℓ steps, the actual expected runtime would be ten times greater than the estimated expected runtime. Thus, the erroneous estimate of that probability would have translated into an unfavorable runtime. This example should illustrate the importance of checking the tails of a distribution for errors as well.

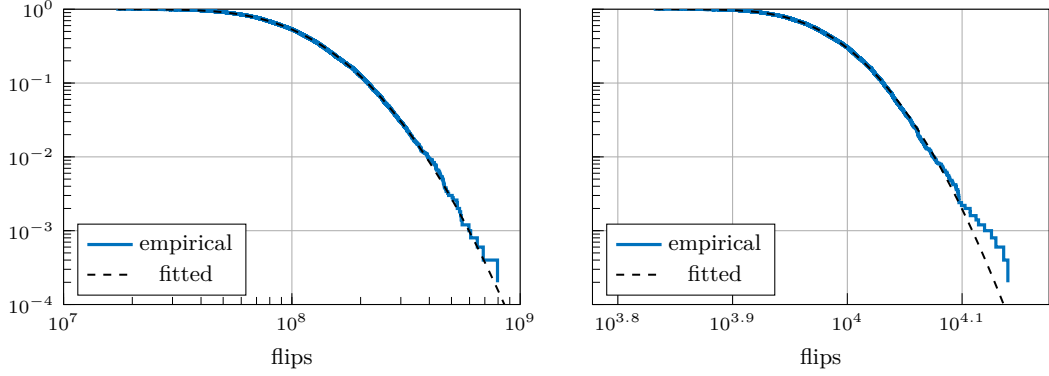
The left tail, i. e., the probabilities for very small values, can be checked visually by plotting the ecdf and fitted cdf with both axes logarithmically scaled. Thereby, the probabilities for extreme events (in this case, especially easy instances) can be measured accurately.



■ **Figure 2** Logarithmically scaled ecdf and fitted cdf of instances *A* (left) and *B* (right).

The two instances *A* and *B* are being examined in this manner in Figure 2. As can be observed, the lognormal fit accurately predicts the probabilities associated with very short runs. For the other instances, lognormal distributions were mostly also able to accurately describe the probabilities for short runs. However, the behavior of the ecdf and the fitted lognormal distribution differed very slightly in a few instances.

Lastly, the probabilities for particularly hard instances should also be checked. Any mistakes in this area could lead to underestimating the likelihood of encountering an exceptionally hard instance. For analyses of this type, the survival function S is a useful tool; if F is the cdf, $S(x) := 1 - F(x)$. Therefore, the survival function's value $S(x)$ represents the probability that an instance is (on average) harder than x in our case. If we plot the empirical survival function, i. e., $\hat{S}_n(x) := 1 - \hat{F}_n(x)$, and the fitted survival function together on a graph with logarithmically scaled axes, we can easily detect errors in the right tail.



■ **Figure 3** Logarithmically scaled empirical survival function and fitted survival function of instances *A* (left) and *B* (right).

Figure 3 illustrates this type of plot for the instances *A* and *B*. Here, there is a discernible deviation between *A* and *B*. While for *A*, the lognormal fit provides an accurate description of the probabilities for long runs, in the case of *B*, the empirical survival function seems to approach 0 somewhat slower than the lognormal estimate. In the vast majority of cases, these extreme value probabilities are accurately reflected by the lognormal fit. In most other cases, the empirical survival function approaches 0 more slowly than the lognormal fit. Thus, in these cases, the likelihood of encountering an exceptionally hard instance is underestimated.

So far, we discussed the behavior of lognormal fits based on this visual inspection. Altogether, we concluded that lognormal distributions seem to be well suited for describing the data. Next, we shall concretize this through a statistical test. To be more precise, we apply the χ^2 -test as a goodness-of-fit test for each base instance. For each such instance, the fitted lognormal distribution used the 5000 data points, and afterwards, the χ^2 -test statistic is computed. Subsequently, the probability that such a value of the test statistic occurs under the assumption of the so-called null hypothesis is determined. We will refer to this probability as the *p*-value. In our case, the null hypothesis is the assumption that the data follow a lognormal distribution. If the fit is poor, then a small *p*-value will occur. If *p* is sufficiently small, the null hypothesis is rejected. We reject the null hypothesis if $p < 0.05$.

Two more remarks are due on this matter. First, from a high *p*-value, one cannot prove that the assumption that the data are lognormally distributed is correct. However, we use a sufficiently high *p*-value as a heuristic whether this assumption is reasonable.

Secondly, there is an obstacle that complicates statistical analysis by this method. As described, each of the 5000 data points is obtained by first sampling 100 runtimes of the corresponding instance and then calculating the mean. This means that we do not work with the actual expected values but only estimates. In other words, this implies that our data is noisy. The greater the variance in the respective instance, the greater the corresponding noise. If one were to apply the χ^2 -test to this noisy data, some cases would be incorrectly rejected, especially if the variance is large. To overcome this limitation, we additionally use a bootstrap-test, which is based on Cheng [9]. This test is presented in Algorithm 3.

Briefly summarized, this test simulates how our data points were generated, assuming the null hypothesis. For this purpose, particular attention should be paid to how the test sample is rendered noisy. Owing to the central limit theorem, it is reasonable to assume that the initial data's sample mean originates from a normal distribution around the true expected value. We use this assumption in the bootstrap-test using a noise signal drawn from a normal distribution with expected value 0. The variance of this normal distribution is determined from the initial data and divided by 100 (cf. central limit theorem).

■ **Algorithm 3** Bootstrap-test for noisy data.

Input: (noisy) random sample $\mathbf{y} = (y_1, y_2, \dots, y_n)$, integer N , significance $\alpha \in (0, 1)$

$\hat{\theta} \leftarrow \text{MLE}(\mathbf{y}, F)$, lognormal maximum likelihood estimation, F is the lognormal cdf

$X^2 \leftarrow \text{ChiSquare}(\mathbf{y}, \hat{\theta})$, Chi-squared goodness of fit test statistic

for $j = 1$ **to** N **do**

$\mathbf{y}' \leftarrow (y'_1, \dots, y'_n)$, where all y'_i are i.i.d. samples from the fitted lognormal distribution with parameters $\hat{\theta}$

$\mathbf{y}' \leftarrow \mathbf{y}' + \text{noise}$, where noise is sampled from an n -dimensional normal distribution

$\hat{\theta}' \leftarrow \text{MLE}(\mathbf{y}', F)$

$X_j^2 \leftarrow \text{ChiSquare}(\mathbf{y}', \hat{\theta}')$

Let $X_{(1)}^2 \leq X_{(2)}^2 \leq \dots \leq X_{(N)}^2$ be the sorted test statistics.

if $X_{(\lfloor (1-\alpha) \cdot N \rfloor)}^2 < X^2$ **then** reject **else** accept;

If there is a large difference between the respective p -values of the χ^2 - and bootstrap-tests, this suggests that the variance in the initial data is too high and that the number of samples used to calculate the sample mean should be increased. However, this case has only occurred twice, and we explicitly indicate it later. In all other cases, the p -values of the χ^2 - and the bootstrap-tests were similar. To illustrate this point, let us consider the p -values of the two instances A and B . The excellent representation of the runtime behavior over the entire support on instance A is also reflected in the two p -values. Using the χ^2 -test, one obtains a p -value of approximately 0.783, and using the bootstrap-test, one obtains a p -value of 0.76. Since these two p -values are both above 0.05, we conclude that the assumption that lognormal distributions describe the runtimes is reasonable.

In contrast, for instance B , we observed that while a lognormal distribution accurately describes the main part of the distribution, the probabilities for extremely long runs are inadequately represented. This observation is again reflected in the respective p -values. The χ^2 -test yields a p -value of ≈ 0.008 , and the bootstrap-test yields a p -value of 0.013. Since both p -values are below 0.05, the assumption that these data originate from a lognormal distribution is rejected.

Overall, this example is intended to demonstrate that these two statistical tests can show an inadequate fit, even if the problems only arise for extreme values. Second, it should demonstrate that the p -values of the two tests are generally similar; if the p -values differ significantly, then more samples should be used to calculate the sample mean.

We now proceed by considering the adequacy of lognormal distributions for describing SRWA runtimes. The results of the statistical analysis are reported in Table 1 and can be found in [42].

■ **Table 1** Statistical goodness-of-fit results for Alfa+SRWA runtimes over various problem domains. The *rejected* row contains the number of instances where the lognormal distribution is not a good fit according to the χ^2 -test at a significance level of 0.05. To put these results into perspective, the second row contains the total number of instances of each domain. Out of a total of 230 instances, 5 got rejected.

	hidden	different chances	uniform	factoring	coloring	total
rejected	0	2	1	2	0	5
# of instances	20	120	25	33	32	230

The first line in the table represents for how many instances the statistical tests rejected the lognormal distribution hypothesis. The second line indicates how many instances have been checked in total. It should be noted that the same number of instances was rejected by the χ^2 -test as was by the bootstrap-test. Thus, there is no need to distinguish between the tests here. It should also be mentioned that in statistical tests, there is always a possibility that a hypothesis will be rejected even though the null hypothesis holds (type 1 error). At a significance level of 0.05, this probability is at the most 5%. Accordingly, the total of 5 rejected instances may be attributed to so-called type 1 errors. This statement is also supported by the fact that no exceptionally low p -value was observed, i. e., no p -value that is unusual for a total of 230 samples.

For **probSAT**, on the other hand, the situation appears to be different. The results are summarized in Table 2 and [42]. The columns refer to the number of variables in the

■ **Table 2** Goodness-of-fit results for **Alfa+probSAT** over various hidden solution instance sizes. The *rejected* row contains the number of instances where the lognormal distribution is not a good fit according to the χ^2 -test at a significance level of 0.05. To put these results into perspective, the second row contains the total number of instances of each instance size.

number of variables	50	100	150	200	300	800	total
rejected	2	2	1	0	2	0	7
# of instances	10	10	10	10	10	5	55

corresponding SAT instances. The number of rejected instances is again identical regardless of whether the χ^2 - or bootstrap-test is applied. As can be seen, the lognormal distribution hypothesis was rejected for 7 of the 55 instances. This number can no longer be accounted for by type 1 errors at a significance level of 0.05. However, one can observe that the majority of rejected instances occur for a small number of variables. If one were to consider only the instances from 150 variables onwards, then the remaining rejected instances may be attributed to type 1 errors. This raises the suspicion that there may be a limiting process, i. e., that the lognormal distribution hypothesis is only valid for $n \rightarrow \infty$.

Lastly, a difference between the two static tests emerges for **YALSAT**. According to the χ^2 -test, 2 of the total 10 instances are rejected. However, using the bootstrap-test, the lognormal distribution hypothesis is not rejected for any instance. Therefore, one cannot rule out the possibility that lognormal distributions are the natural model to describe the instances, but more experiments are required to make a more precise statement.

In summary, the presumption that lognormal distributions are the appropriate choice for describing runtimes has been reinforced for **SRWA**. For **probSAT**, this appears plausible at least above a certain instance size. Likewise, the choice of lognormal distributions also seems reasonable for **YALSAT**. These observations lead us to the following conjecture.

► **Conjecture 6** (Strong Conjecture). *The runtime of **Alfa** with $SLS \in \{\text{SRWA}, \text{probSAT}, \text{YalSAT}\}$ follows a lognormal distribution.*

If this statement is true, then it would be intriguing in that one can infer how modifying the base instance affects the hardness of instances. This effect is likely to be the result of generating models for lognormal distributions. Just as the normal distribution is a natural model for the sum of i. i. d. random variables, the lognormal distribution is a natural model for the multiplication of i. i. d. random variables. Thus, one can hypothesize that each added clause exerts a small multiplicative effect on the instance's hardness.

Simultaneously, the three parameters of the lognormal distribution also provide insight into how the hardness of the instance changes. For example, the location parameter γ implies an inherent problem hardness that cannot be decreased regardless of the added clauses'

choice. At the same time, γ also serves as a numerical description for the value of this intrinsic hardness. Using Bayesian statistics, it is possible to infer the parameters while the solver is running. These estimated parameters can, for example, be used to schedule restarts. This would lead to a scenario similar to that discussed in [34].

Conjecture 6 is a strong statement. However, a small deviation of the probabilities, for example, at the left tail, would render the strong conjecture invalid from a strict mathematical point of view. Particularly, visual analyses revealed that the left tail's behavior, i. e., for extremely short runs, is occasionally not accurately reflected by lognormal distributions. Conversely, the right tail, i. e., the probabilities for particularly long runs, are usually either correctly represented by lognormal distributions or, occasionally, the corresponding probability approaches 0 even more slowly. We, therefore, rephrase our conjecture in a weakened form. Our observations fit a class of distributions known as long-tail distributions defined purely in terms of their behavior at the right tail.

► **Definition 7** ([12]). *A positive, real-valued random variable X is long-tailed, if and only if*

$$\forall x \in \mathbb{R}^+ : \Pr[X > x] > 0 \quad \text{and} \quad \forall y \in \mathbb{R}^+ : \lim_{x \rightarrow \infty} \frac{\Pr[X > x + y]}{\Pr[X > x]} = 1.$$

► **Conjecture 8** (Weak Conjecture). *The runtime of **Alfa** with $SLS \in \{SRWA, probSAT, YaSAT\}$ follows a long-tailed distribution.*

It should be noted that lognormal distributions have the long-tail property [12, 30]. That is, if the Strong Conjecture holds, the Weak Conjecture is implied. The reverse is, however, not true. In the next section, we show an important consequence in case the Weak Conjecture holds.

5 Restarts Are Useful For Long-Tailed Distributions

If the Strong Conjecture holds, i. e., if the runtimes are lognormally distributed, then restarts are useful [22]. This section extends this result and mathematically proves that restarts are useful even if only the Weak Conjecture holds. This will be achieved by showing that restarts are useful for long-tailed distributions.

A condition for the usefulness of restarts, as defined in Definition 1, was proven in [22]. We will show the result using this theorem that is restated below.

► **Theorem 9** ([22]). *Let X be a positive, real-valued random variable having quantile function Q , then restarts are useful if and only if there is a quantile $p \in (0, 1)$ such that*

$$R(p, X) := (1 - p) \cdot \frac{Q(p)}{E[X]} + \frac{\int_0^p Q(u) du}{E[X]} < p.$$

Even if the quantile function and the expected value are unknown, $R(p, X)$ can be characterized for large values of p .

► **Lemma 10.** *Consider a positive, real-valued random variable X with pdf f and quantile function Q such that $E[X] < \infty$. Also, assume that the limit $\lim_{t \rightarrow \infty} t^2 \cdot f(t)$ exists. Then,*

$$\lim_{p \rightarrow 1} R(p, X) = \lim_{p \rightarrow 1} \left((1 - p) \cdot \frac{Q(p)}{E[X]} + \frac{\int_0^p Q(u) du}{E[X]} \right) = 1.$$

Proof. In the following, let F and f be the cdf and pdf of X , respectively. We start by specifying the derivative of Q with respect to p as a preliminary consideration. From $F = Q^{-1}$ and the application of the inverse function theorem [35] it follows:

$$Q'(p) := \frac{d}{dp} Q(p) = \frac{1}{f(Q(p))}. \quad (1)$$

As the first step in our proof, we consider the limiting value of the second summand of $R(p, X)$. This value can be determined by integration by substitution with $x = Q(u)$ followed by applying the change of variable method with $p = F(t)$:

$$\lim_{p \rightarrow 1} \frac{\int_0^p Q(u) du}{E[X]} = \lim_{p \rightarrow 1} \frac{\int_0^{Q(p)} x \cdot f(x) dx}{E[X]} = \lim_{t \rightarrow \infty} \frac{\int_0^t x \cdot f(x) dx}{E[X]} = 1.$$

The last equality holds because the numerator matches the definition of the expected value.

Next, we examine the limit of $(1-p)Q(p)/E[X]$. Since $\lim_{p \rightarrow 1}(1-p) = 0$, the limit of $(1-p) \cdot Q(p)/E[X]$ needs to be examined more closely. For this purpose, L'Hospital's rule is applied twice as well as the change of variable method with $p = F(t)$ is used in the following:

$$\lim_{p \rightarrow 1} (1-p) \cdot Q(p) = \lim_{p \rightarrow 1} Q(p)^2 \cdot f(Q(p)) = \lim_{t \rightarrow \infty} t^2 \cdot f(t).$$

It is well-known that if $\liminf_{t \rightarrow \infty} t^2 \cdot f(t) > 0$ were to hold, then the expected value $E[X]$ would be infinite (this statement is, for example, implicitly given in [12]). This would contradict the premise of the lemma; therefore, $\liminf_{t \rightarrow \infty} t^2 \cdot f(t) = 0$. Moreover, since, by assumption, $\lim_{t \rightarrow \infty} t^2 \cdot f(t)$ exists, we may conclude that

$$\lim_{t \rightarrow \infty} t^2 \cdot f(t) = \limsup_{t \rightarrow \infty} t^2 \cdot f(t) = \liminf_{t \rightarrow \infty} t^2 \cdot f(t) = 0. \quad \blacktriangleleft$$

A frequently used tool for the description of distributions is the hazard rate function.

► **Definition 11** ([31]). *Let X be a positive, real-valued random variable having cdf F and pdf f . The hazard rate function $r: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ of X is given by*

$$r(t) := \frac{f(t)}{1 - F(t)}.$$

In particular, there is an interesting relationship between the long-tail property and the hazard rate function's behavior.

► **Lemma 12** ([30]). *Let X be a positive, real-valued random variable with hazard rate function r such that the limit $\lim_{t \rightarrow \infty} r(t)$ exists. Then, the following three statements are equivalent:*

1. X is long-tailed.
2. $\lim_{x \rightarrow \infty} \int_x^{x+y} r(t) dt = 0, \quad \forall y > 0.$
3. $\lim_{t \rightarrow \infty} r(t) = 0.$

Proof. The full length-version of this paper [41] contains a proof of this lemma since the manuscript [30] was still unpublished at the time of writing. ◀

With the help of these preliminary considerations, we are now ready to show that restarts are useful for long-tailed distributions.

► **Theorem 13.** *Consider a positive, long-tailed random variable X with continuous pdf f and hazard rate function r . Also assume that either $E[X] = \infty$ holds or the limits $\lim_{t \rightarrow \infty} r(t)$ and $\lim_{t \rightarrow \infty} t^2 \cdot f(t)$ both exist. In both cases, restarts are useful for X .*

Proof. Let F be the cdf and Q the quantile function of X . We begin with the case $E[X] = \infty$. According to Theorem 9, restarts are useful if and only if

$$(1-p) \cdot \frac{Q(p)}{E[X]} + \frac{1}{E[X]} \cdot \int_0^p Q(u) du < p$$

for some $p \in (0, 1)$. However, if the expected value $E[X]$ is infinite, then the left side of this inequality is zero and the inequality is obviously satisfied. Hence, the statement follows.

Secondly, we assume that $E[X] < \infty$ and that both $\lim_{t \rightarrow \infty} r(t)$ and $\lim_{t \rightarrow \infty} t^2 \cdot f(t)$ exist. Equation (1) can now be used to calculate the following derivative:

$$\frac{d}{dp} (R(p, X) - p) = \frac{d}{dp} \left((1-p) \cdot \frac{Q(p)}{E[X]} + \frac{\int_0^p Q(u) du}{E[X]} - p \right) = \frac{1-p}{E[X] \cdot f(Q(p))} - 1.$$

Consider the limit of this expression for $p \rightarrow 1$. Once again, the change of variable method is applied with $p = F(t)$, resulting in:

$$\lim_{p \rightarrow 1} \frac{1-p}{E[X] \cdot f(Q(p))} - 1 = \lim_{t \rightarrow \infty} \frac{1-F(t)}{E[X] \cdot f(t)} - 1 = \lim_{t \rightarrow \infty} \frac{1}{E[X] \cdot r(t)} - 1.$$

By assumption, X has a long-tail distribution and the limit of $\lim_{t \rightarrow \infty} r(t)$ exists. For this reason, $\lim_{t \rightarrow \infty} r(t) = 0$ follows as a result of Lemma 12. Furthermore, since $E[X] < \infty$ holds, we may conclude that

$$\lim_{p \rightarrow 1} \frac{1-p}{E[X] \cdot f(Q(p))} - 1 = \lim_{t \rightarrow \infty} \frac{1}{E[X] \cdot r(t)} - 1 = \infty. \quad (2)$$

The condition from Theorem 9 can be rephrased in such a way that restarts are useful if and only if $R(p, X) - p < 0$. According to Lemma 10, the left-hand side of this inequality approaches 0 for $p \rightarrow 1$. However, as has been shown in Equation (2), the derivative of $R(p, X) - p$ approaches infinity for $p \rightarrow 1$. These two observations imply that there is a $p \in (0, 1)$ satisfying $R(p, X) - p < 0$. Consequently, restarts are useful for X . ◀

It should be noted that the conditions of this theorem are not restrictive since all naturally occurring long-tail distributions satisfy these conditions (see also [30]).

► **Conjecture 14** (Corollary of the Weak Conjecture). *Restart are useful for Alfa with $SLS \in \{SRWA, probSAT, YaLSAT\}$.*

If Conjecture 8 is true, then this statement follows immediately by Theorem 13.

6 Conclusion

We have provided compelling evidence that the runtime of **Alfa** follows a long-tailed or lognormal distribution. According to [38], the usefulness of restarts is a necessary, however not a sufficient, condition to obtain super-linear speedups by parallelization. Since we have shown that the necessary condition is (presumably) satisfied, this immediately raises the question of whether super-linear speedups are obtained by parallelizing **Alfa**-type algorithms.

We additionally want to pose the question whether some of the Conjectures 6 or 8 can be theoretically proven. A first line of attack would be to analyze the special case of an solver like **SRWA** whose runtime was already theoretically analyzed.

The technique of analyzing the runtime distribution of **Alfa** could be further developed to help better understand the behavior of CDCL solvers. These kind of solvers heavily employ the technique of adding new clauses and deleting some clauses. This can be thought of as solving a new logically equivalent formula of the base instance.

Preliminary results on the solvers excluded for heuristic reasons seem to suggest that the **Alfa**-method forces the runtime of the base solver to exhibit a multimodal behavior. Thus, the lognormal distribution is not a good fit in this case. However, an initial visual inspection of the data indicates an even heavier tail.

References

- 1 Alejandro Arbelaez, Charlotte Truchet, and Philippe Codognet. Using sequential runtime distributions for the parallel speedup prediction of SAT local search. *Theory and Practice of Logic Programming*, 13(4-5):625–639, 2013.
- 2 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, 2009.
- 3 Adrian Balint. Original implementation of *probSAT*, 2015. Available at <https://github.com/adrianopolus/probSAT>.
- 4 Adrian Balint, Anton Belov, Marijn J. H. Heule, and Matti Järvisalo. SAT Competition 2013: Results. URL: <http://satcompetition.org/2013/results.shtml>.
- 5 Tomáš Balyo and Lukáš Chrpá. Using algorithm configuration tools to generate hard SAT benchmarks. In *Proceedings of the 11th International Symposium on Combinatorial Search (SOCS '18)*, pages 133–137. AAAI Press, 2018.
- 6 Wolfgang Barthel, Alexander K. Hartmann, Michele Leone, Federico Ricci-Tersenghi, Martin Weigt, and Riccardo Zecchina. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Physical review letters*, 88(188701):1–4, 2002.
- 7 Armin Biere. Yet another local search solver and Lingeling and friends entering the SAT Competition 2014. In *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*, volume B-2014-2, pages 39–40. University of Helsinki, 2014.
- 8 Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.
- 9 Russell Cheng. *Non-standard parametric statistical inference*. Oxford University Press, 2017.
- 10 Maximilian Diemer. Source code of *GenFactorSat*, 2021. Newest version available at <https://github.com/madiemer/gen-factor-sat/>.
- 11 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Selected Revised Papers of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.
- 12 Sergey Foss, Dmitry Korshunov, and Stan Zachary. *An Introduction to Heavy-Tailed and Subexponential Distributions*, volume 6. Springer, 2011.
- 13 Daniel Frost, Irina Rish, and Lluís Vila. Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI '97)*, pages 327–333, 1997.
- 14 Oliver Gableske. Source code of *kcnfgen (version 1.0)*, 2015. Retrieved from https://www.gableske.net/downloads/kcnfgen_v1.0.tar.gz.
- 15 Carla P. Gomes and Bart Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI '97)*, pages 190–197, 1997.

- 16 Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000. Related version in *CP '97*.
- 17 Marijn J. H. Heule, Matti Järvisalo, and Tomáš Balyo. SAT Competition 2017: Results. URL: <https://baldur.iti.kit.edu/sat-competition-2017/index.php?cat=results>.
- 18 Holger H. Hoos and Thomas Stützle. Evaluating Las Vegas algorithms: Pitfalls and remedies. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*, pages 238–245, 1998.
- 19 Norman L. Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous Univariate Distributions, Volume 1*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2nd edition, 1994.
- 20 Alexis C. Kaporis, Lefteris M. Kirousis, and Yannis C. Stamatiou. A note on the non-colorability threshold of a random graph. *The Electronic Journal of Combinatorics*, 7(1), 2000.
- 21 Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: A generator of crafted benchmarks. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, pages 464–473, 2017.
- 22 Jan-Hendrik Lorenz. Runtime distributions and criteria for restarts. In *Proceedings of the 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '18)*, pages 493–507. Springer, 2018.
- 23 Jan-Hendrik Lorenz and Florian Wörz. On the effect of learned clauses on stochastic local search. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 89–106. Springer, 2020. Implementation and statistical tests of GapSAT available at Zenodo doi:10.5281/zenodo.3776052.
- 24 Jan-Hendrik Lorenz and Florian Wörz. Source code of *concealSATgen*, 2021. Newest version available at <https://github.com/FlorianWoerz/concealSATgen/>.
- 25 Chuan Luo, Shaowei Cai, and Kaile Su. DCCAlm in SAT Competition 2016. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*, volume B-2016-1. University of Helsinki, 2016.
- 26 Chuan Luo, Shaowei Cai, Wei Wu, and Kaile Su. Focused random walk with configuration checking and break minimum for satisfiability. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP '13)*, volume 8124 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 2013.
- 27 Chuan Luo, Shaowei Cai, Wei Wu, and Kaile Su. Double configuration checking in stochastic local search for satisfiability. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2703–2709. AAAI Press, 2014.
- 28 Chuan Luo, Shaowei Cai, Wei Wu, and Kaile Su. CSCCSat in SAT Competition 2016. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*, volume B-2016-1. University of Helsinki, 2016.
- 29 Stephan Mertens, Marc Mézard, and Riccardo Zecchina. Threshold values of random k -SAT from the cavity method. *Random Structures & Algorithms*, 28(3):340–373, 2006.
- 30 Jayakrishnan Nair, Adam Wierman, and Bert Zwart. The fundamentals of heavy tails: Properties, emergence, and estimation. Preprint, California Institute of Technology, 2020.
- 31 Marvin Rausand, Anne Barros, and Arnljot Hoyland. *System reliability theory: models, statistical methods, and applications*. John Wiley & Sons, 2nd edition, 2003.
- 32 Irina Rish and Daniel Frost. Statistical analysis of backtracking on inconsistent CSPs. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP '97)*, pages 150–162, 1997.
- 33 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

- 34 Yongshao Ruan, Eric Horvitz, and Henry A. Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP '02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 573–586. Springer, 2002.
- 35 Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.
- 36 Uwe Schöning. A probabilistic algorithm for k -SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002. Preliminary version in *FOCS '99*.
- 37 Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*, volume 3 of *Mathematics for Applications (Mathematik für Anwendungen)*. Lehmanns Media, 2013.
- 38 Oleg V. Shylo, Timothy Middelkoop, and Panos M. Pardalos. Restart strategies in optimization: parallel and serial cases. *Parallel Computing*, 37(1):60–68, 2011.
- 39 Gunnar Völkel, Ludwig Lausser, Florian Schmid, Johann M. Kraus, and Hans A. Kestler. Sputnik: *ad hoc* distributed computation. *Bioinformatics*, 31(8):1298–1301, 2015.
- 40 Sven Dag Wicksell. On logarithmic correlation with an application to the distribution of ages at first marriage. *Meddelanden från Lunds Astronomiska Observatorium*, 84:1–21, 1917.
- 41 Florian Wörz and Jan-Hendrik Lorenz. Evidence for long-tails in SLS algorithms. Technical Report 2107.00378, arXiv.org, 2021. This is the full-length version of the present paper.
- 42 Florian Wörz and Jan-Hendrik Lorenz. Supplementary Data for “Evidence for Long-Tails in SLS Algorithms”, 2021. We have provided all data of this paper. All base instances, resolvents, and modifications can be found under doi:10.5281/zenodo.4715893. Visual and statistical evaluations can be found under https://github.com/FlorianWoerz/SLS_Evidence_long_tail, where all evaluations take place in the files `./evaluation/jupyter/evaluate_*.ipynb`. A permanent version of this repository has been preserved under doi:10.5281/zenodo.5026180.